TURING 图灵程序设计丛书

[PACKT] PUBLISHING

[巴西] Loiane Groner 著　卢俊祥 译

精通Ext JS

Mastering Ext JS

- 花旗银行软件开发经理、IBM系统分析师8年工作经验总结
- 汇聚Java用户组领导者的精粹技巧
- 每章一个任务，分模块细析应用开发



人民邮电出版社
POSTS & TELECOM PRESS

数字版权声明

图灵社区的电子书没有采用专有客户端，您可以在任意设备上，用自己喜欢的浏览器和PDF阅读器进行阅读。

但您购买的电子书仅供您个人使用，未经授权，不得进行传播。

我们愿意相信读者具有这样的良知和觉悟，与我们共同保护知识产权。

如果购买者有侵权行为，我们可能对该用户实施包括但不限于关闭该帐号等维权措施，并可能追究法律责任。



Loiane Groner

花旗银行软件开发经理，负责海外项目的开发和团队管理工作；原IBM公司系统分析师及团队负责人；巴西坎皮纳斯Java用户组（CampinasJUG/SouJava Campinas）领导者、圣埃斯皮里图Java用户组（ESJUG）协调人；巴西各大型技术会议特邀发言人；Sencha和Java技术布道者，通过博客（http://loianegroner.com和http://loiane.com）为软件开发社区撰稿，发表关于IT职业发展、Ext JS、Sencha Touch、Sencha Architect、Java及常用开发技术方面的文章和视频。另外，她还著有*Ext JS 4 First Look*和*Sencha Architect App Development*。

图灵程序设计丛书

[巴西] Loiane Groner 著　卢俊祥 译

精通Ext JS

Mastering Ext JS

人民邮电出版社
北京

图书在版编目（C I P）数据

精通Ext JS / (巴西) 格罗纳 (Groner,L.) 著 ; 卢俊祥译. -- 北京 : 人民邮电出版社, 2014.3
（图灵程序设计丛书）
书名原文: Mastering Ext JS
ISBN 978-7-115-34723-7

Ⅰ. ①精… Ⅱ. ①格… ②卢… Ⅲ. ①JAVA语言一程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2014)第031250号

内 容 提 要

《精通 Ext JS》站在开发者的角度思考问题，用实际示例讲解如何用 Ext JS 组件实现绝妙的应用程序，并展现从界面原型到产品化构造的各个阶段，最终实现一个完整的应用程序。Loiane Groner 将带我们构建应用结构、启动界面、登录界面、多语言支持功能、行为监控功能、取决于用户权限的动态菜单，以及（或简单或复杂的）数据库信息管理模块。之后，我们会学习产品构造方法、将 Web 应用转换成原生桌面应用，以及调试与测试。本书后面还专设一章，介绍如何使用 Ext JS 创建 WordPress 主题。

本书适合 Ext JS 开发人员，以及欲进一步提升技能开发更优秀 Web 应用的开发人员阅读参考。

◆ 著　　[巴西] Loiane Groner
译　　卢俊祥
责任编辑　毛倩倩
执行编辑　张　庆
责任印制　焦志炜

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　　　　　　　　　印刷

◆ 开本：800×1000　1/16
印张：16.75
字数：396千字　　　　　2014年 3 月第 1 版
印数：1 – 4 000册　　　2014年 3 月北京第 1 次印刷
著作权合同登记号　图字：01-2013-6735号

定价：59.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316

反盗版热线：(010)81055315

广告经营许可证：京崇工商广字第 0021 号

版 权 声 明

Copyright © 2013 Packt Publishing. First published in the English language under the title *Mastering Ext JS*.

Simplified Chinese-language edition copyright © 2014 by Posts & Telecom Press. All rights reserved.

本书中文简体字版由Packt Publishing授权人民邮电出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

版权所有，侵权必究。

译　者　序

任何技术的发展都伴有或褒或贬的声音，Ext JS也不可避免。记得在2007、2008年的时候，各大前端论坛对Ext JS和jQuery的争论很多，许多人都希望自己钟爱的技术能够“一统江湖”。现在回头想想，码农对技术的争论永远是一道亮丽的风景线。

当时对Ext JS的争论主要集中在两点上：一是Ext JS很酷，开发效率很高，UI效果拿来即用，比jQuery方便；二是认为Ext JS太“重”，不适合互联网应用，还是jQuery轻巧。时至今日，这两种论调已不多见，开发者的心态及认识也逐渐回归理性。是的，每种技术都有其合适的应用场景。“天生我材必有用”，多年过去了，这两种技术谁也没有取代谁，恰恰相反，它们都在各自的道路上发展得很好、很滋润。如今，Ext JS和jQuery都已延伸到了移动端，相应的开发者规模也越来越庞大，这足以让我感慨万千。

正如Sencha.com网站上介绍的，Ext JS是一个基于JavaScript的企业级Web应用开发框架，当前最新版本是4.2.*x*。这里要着重强调一点：Ext JS 4相对于之前的版本变化巨大，而且是核心架构级的改进，包括类系统、MVC应用架构、渲染和布局等多方面。可以说，Ext JS 4是Ext JS 2之后又一个里程碑式的版本。

《精通Ext JS》是一本比较独特的书，它没有机械地罗列Ext JS各个组件。同时，它也不是一本有关Ext JS的入门书籍（但我觉得门槛并不高）。全程通过一个完整的例子，按开发流程递进式讲解相应知识点，其间穿插着大量最佳实践。我认为这是本书最大的亮点。毫无疑问，本书跟动不动就罗列一大堆API的书划清了界限。我更喜欢这本书。

翻译完毕，书中错误难免，心里感觉忐忑，还请各位读者适度拍砖。但能遇到一本好书，又是件高兴的事情。翻译过程中，承蒙图灵编辑李松峰、朱巍、毛倩倩、张庆，以及业内同行贾洪峰、许晓斌、陈睿杰等各位老师的无私指导，在此特致谢意！

www.packtpub.com

支持文件、电子书及打折优惠

欢迎读者访问www.packtpub.com下载支持文件及其他相关资源。

Packt为所有已出版图书提供电子版（PDF和ePub文件）。你可以在www.packtpub.com上升级电子书，且纸质书的用户有权享受购买其电子版的折扣。要了解更多信息，请发邮件到service@packtpub.com。

在www.packtpub.com还可以阅读免费的技术文章，通过注册收取一系列最新促销信息，并获得Packt纸质书及电子书的独家折扣或享受特价购书。



http://PacktLib.PacktPub.com

你是否经常受到一些IT难题的困扰？PacktLib是Packt的在线数字图书馆。在这里，你可以访问、阅读和搜索整个Packt图书馆里的图书。

为什么订阅

- 可以搜索Packt出版的所有图书；
- 可以复制、粘贴、打印内容并在内容上添加书签；
- 可以通过浏览器实现按需访问。

Packt 注册用户大礼

在www.packtpub.com上注册一个Packt账号，就可以立即访问PacktLib并自由浏览9本图书。注册 → 登录 → 浏览，就这么简单。

技术审校

Aafrin是一位自学成才的程序员，具有网络安全及数字取证方面的专业背景。他从2003年起一直从事Web应用程序的设计和开发工作，使用过C++、Java、PHP、ASP、VB、VB.NET等多种编程语言，以及Ext JS、CakePHP、CodeIgniter、Yii等框架。业余时间，他在www.aafrin.com上写博客，并从事计算机安全/计算机取证领域的研究。

Vincenzo Ampolo在意大利米兰理工大学获得了计算机系统工程硕士学位，并有六年多的自由职业者经历。9岁时，他就对技术产生了浓厚兴趣，并在那年组装了第一台属于自己的计算机。他12岁时学习了BASIC，15岁时已经掌握了C，之后又学习了Python和JavaScript。他是自由软件传播者，并且从14岁就开始使用GNU/Linux。17岁时，他开发了第一个Linux用户模式启动界面。一年前他来到了硅谷。

他还是个热心人，他相信软件社区的力量，并帮助组织了许多FOSS（Free and Open Source Software，自由软件和开源软件）相关的会议，如Linux day和Ubuntu release party。

献给所有相信我的人。

Yiyu Jia从1996年就致力于开发Web应用。他参与了很多以Java和PHP为后台语言的项目，在其中担任技术主管和系统架构师。同时，他还有互动电视、中间件和家庭网关等方面的工作经验。他对跨平台Web应用的设计尤其感兴趣。

同时他还是新的数据挖掘研究课题——增进式子空间挖掘（Promotional Subspace Mining，PSM）的主要创始人，该技术的目标是在巨大数据集的子空间中发掘有用信息。他现在从事大数据技术研究。

你可以通过他的博客和网站找到他：http://yiyujia.blogspot.com及http://www.idatamining.org。

Joel Watson是一名Web爱好者，在过去的八年里一直从事网站设计和开发的工作。他喜欢研究各种各样的Web技术，并特别喜欢利用HTML5及相关技术的最新特性来创建具有很棒Web体验的应用。

不写代码的时候，Joel喜欢跟他的妻子和两个女儿待在一起，弹吉他或者看精彩的科幻片和动画片。

前　　言

作为Ext JS开发人员，你很可能已经花费了一些时间来学习这个框架。我们都知道，Ext JS并非是一朝一夕就可以掌握的。学会Ext JS的基本应用后，我们需要在日常开发中使用它，此时大量问题也就冒了出来。两个组件间如何通信？最佳做法是什么？为什么选择这种方式，而不是其他方式？是否还有实现同一种特性的其他方式？这都是普遍存在的问题。

本书站在开发者的角度思考问题：如何融合各种思路，通过Ext JS创建绝妙的应用呢？

这正是本书要讲解的内容。我们将经历从界面原型到产品化构造的各个阶段，最终实现一个完整的应用程序。作者将带我们构建应用结构、启动界面、登录界面、多语言支持功能、行为监控功能、取决于用户权限的动态菜单，以及（简单或复杂的）数据库信息管理模块。之后，我们还将学习产品构造方法、如何将Web应用转换成原生桌面应用，以及怎样调试、测试它。本书后面还专设一章，介绍如何使用Ext JS创建WordPress主题。

本书将使用实际示例，并讲解怎样用Ext JS组件来实现它们。我们还将给出大量提示、选择具体做法的原因与依据，以及各种最佳实践，帮助大家把Ext JS技能提升到一个新的高度。

本书内容

第1章介绍本书实现的应用示例及其特性、每个界面和模块的原型（后续每一章介绍一个模块），并讨论了如何用MVC架构的思路创建Ext JS应用程序，以及如何创建启动界面。

第2章讨论如何使用Ext JS实现登录界面，以及怎样在服务器端做相应处理，还将介绍各种增强功能的实现：大写键提醒、回车键提交表单和密码发送到服务器端之前对其进行加密。

第3章介绍注销、客户端行为超时监控（如果用户一段时间内未使用鼠标或键盘，系统将自动终止会话并注销）等功能。这一章还将介绍多语言支持，并为此创建一个更改系统语言和本地化设置的组件。

第4章创建一个取决于用户权限的动态菜单。根据用户权限情况来决定菜单项如何渲染呈现，如果用户权限不足，就不会显示菜单项。

第5章讨论用户管理与安全，将创建两个功能界面：列出所有系统用户的操作界面，以及添加/编辑用户、删除现有用户和更改用户权限的操作界面。

第6章实现用户编辑数据库表信息的模块，其界面看起来与MySQL数据库表编辑器很相似。同时还将介绍即席搜索、过滤、行内编辑（使用单元格编辑插件）等。这一章也将讨论实际开发Ext JS大型应用程序时的一些话题，如组件重用。

第7章进一步探讨数据库表信息管理以及表与表之间的关联，内容涉及管理复杂信息、处理数据网格与表单面板间的关系。

第8章介绍（Ext JS原生不支持的）打印、导出PDF和Excel等功能，还将介绍图表和将其导出成图片和PDF格式的方法，以及怎样使用第三方插件。

第9章会实现一个与微软Outlook非常相似的电子邮件客户端界面，但只讨论用Ext JS创建相应界面，不涉及使用邮件功能库收发邮件。

第10章简单地介绍Ext JS 4.2自定义主题，并讨论产品构造的步骤和好处，以及如何通过Sencha Desktop Packager创建Ext JS原生桌面应用。

第11章与本书其他各章的主线内容不同，讨论如何通过Ext JS创建WordPress主题，探讨的是Ext JS的不同应用场景。

第12章讨论调试Ext JS应用程序、注意事项以及掌握调试方法的重要性。这一章还将简单介绍用于测试Ext JS应用程序的Siesta框架，以及转换Ext JS应用为移动应用的思路。最后，还将推荐一些便于日常开发任务的有用工具，并告诉你去哪儿寻找可在Ext JS项目中使用的第三方插件。

阅读须知

以下是运行本书示例需预装的软件，当然你也可以使用已经安装的同类软件。

带有调试工具的浏览器：

- ❑ Firefox及Firebug　https://www.mozilla.org/firefox/和 http://getfirebug.com/
- ❑ Chrome　http://www.google.com/chrome

PHP Web服务器软件：

- ❑ XAMPP　http://www.apachefriends.org/en/xampp.html

数据库：

- ❑ MySQL　http://dev.mysql.com/downloads/mysql/

- MySQL Workbench　http://dev.mysql.com/downloads/tools/workbench/
- MySQL Sakila样例数据库　http://dev.mysql.com/doc/index-other.html和http://dev.mysql.com/doc/sakila/en/index.html

Sencha Command及其他工具：

- Sencha Command　http://www.sencha.com/products/sencha-cmd/download
- Ruby　http://www.ruby-lang.org/en/downloads/
- Sass　http://sass-lang.com/
- Compass　http://compass-style.org/
- Java JDK　http://www.oracle.com/technetwork/java/javase/downloads/index.html
- Java环境变量设置　http://docs.oracle.com/javase/tutorial/essential/environment/paths.html
- Ext JS　http://www.sencha.com/products/extjs/

本书使用Ext JS 4.2。

读者对象

本书适合Ext JS专业开发人员，以及那些想进一步提升技能以开发出更优秀Web应用的开发人员。本书并不讲述Ext JS基础知识。

排版约定

为方便读者阅读，本书对不同信息采用不同样式。接下来，我们来看样式的示例和解释。

正文中提到的代码格式如下所示：“我们将为加载的`DIV`标签添加新的CSS样式。”

代码段样式如下：

```
Ext.application({ // #1
    name: 'Packt', // #2
    launch: function() { // #3
         console.log('launch');   // #4
    }
});
```

当需要读者特别注意代码块中的某一部分时，相关代码行或项将以粗体表示：

```
controllers: [
    'Login',
    'TranslationManager',
    'Menu'
]
```

命令行输入与输出样式如下：

```
sencha generate theme masteringextjs-theme
```

新术语及重要词汇将采用楷体字。



警告或重要说明将写在这里。



小贴士和技巧将写在这里。

读者反馈

我们时刻欢迎你的反馈，以便了解你对本书的看法。你的宝贵意见有助于我们提升书籍的质量。

一般的阅读反馈，可直接发送电子邮件至feedback@packtpub.com，请在邮件标题中注明书名。

如果你在某个领域内有专长且有兴趣编写相关书籍，请访问www.packtpub.com/authors查看作者指南。

客户支持

现在你已是Packt图书的尊贵读者了，我们有一系列的售后支持，保证你的消费物有所值。

代码下载

访问下面的网址获取本书示例的源代码：https://github.com/loiane/masteringextjs。

勘误[1]

尽管我们已经对书籍作了仔细校对以保证内容准确，但错误在所难免。如果在书中发现任何的文字或代码错误，非常欢迎你将这些错误提交给我们，这样可以帮助我们在后续版本中改正错误，避免其他读者产生不必要的误解。一旦发现错误，请登录http://www.packtpub.com/support，

① 中文版读者可免费注册iTuring.cn在本书页面（~/book/1189）提交勘误。——编者注

选择书名，点击errata submission form（提交勘误）链接，然后填写具体的错误信息即可。只要你提交的勘误通过验证，勘误信息就会上传到我们的网站，或者追加到已有勘误列表中，显示在该书的勘误页面。

盗版

对所有媒体来说，互联网盗版都是一个棘手的问题。Packt很重视版权保护。如果你在互联网上发现我们公司出版物的任何非法复制品，请及时告知我们相关网址或网站名称，以便我们采取补救措施。

如果发现可疑盗版材料，请通过copyright@packtpub.com联系我们。

对你帮助我们保护作者权益、确保我们持续提供高品质图书的行为表示敬意。

其他

如果你对本书有任何问题，请通过questions@packtpub.com联系我们，我们会尽力解决。

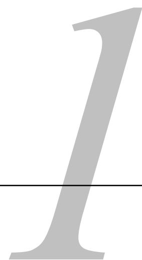
致　　谢

感谢父母给我的教育、指导和建议，在我的成长岁月里，帮助我成为一位优秀的专业技术人员。尤其要感谢我的丈夫，感谢你的耐心、支持以及对我的莫大鼓励。同时，非常感谢我的朋友以及一直支持我的读者。

目　　录

第 1 章

启　　程



Ext JS是知名的跨浏览器RIA（Rich Internet Application，富因特网应用）框架，用来创建丰富且用户友好的Web前端界面。通过研究Ext JS SDK里的示例，你能够知晓如何使用Grid、Tree、Form和Chart等组件，以及MVC（Model-View-Controller，模型−视图−控制器）架构。但这些示例大多数都是彼此无关的，因此很难把它们集成到一个应用里。此外，在开发应用时，某些情况下通常可以重用大量代码，从而使代码易于维护。

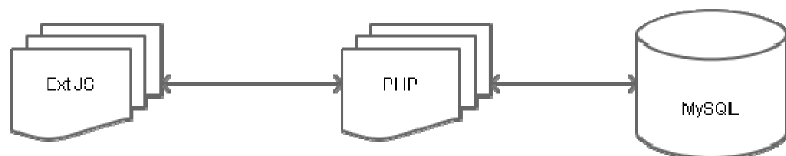
在本书中，我们将畅游Sencha Ext JS的世界，并研究实际案例，开发一个完整的应用，从无到有，从原型阶段直至产品部署。

本章将介绍要开发的应用，学习如何组织将在各章创建的应用文件。同时，本章也会展示应用原型（mockup），阐述如何开始组织界面（这一点很重要，但一些开发者常常忘记组织界面）。本章主要内容如下：

- 安装所需软件；
- 展示应用及其功能；
- 创建每个界面的原型；
- 用MVC方式创建应用的结构；
- 创建加载页面。

1.1　安装所需软件

我们要开发的应用有一个比较简单的架构，前端采用Ext JS，它与服务器端模块通信，服务器端模块与数据库通信，整体架构如下图所示。



服务器端模块的构建使用PHP。如果你不懂PHP也不必太担心，因为我们只用非常简单的代码，而且重点关注必须在服务器端实现的逻辑。同样的逻辑也可以用其他提供JSON或XML数据的编程语言来实现，比如Java、ASP.NET、Ruby、Python，等等。Ext JS使用JSON或XML作为数据通信格式。

数据库选用MySQL。同时，使用Sakila样例数据库，它很适合用来演示数据库表的CRUD（Create-Read-Update-Delete/Destroy，创建–读取–更新–删除/销毁）操作以及其他复杂操作，比如视图和存储过程（后续将学习怎样结合Ext JS做这些处理）。

应用实现之后，我们还需要定制主题，因此，需要安装Ruby、Sass和Compass的gem包。另外，定制主题、构建系统也需要安装Sencha Command。为了保证Sencha Command能够正确工作，我们还需要安装并设置Java SDK。

部署应用需要Web服务器软件。如果你的电脑里没有安装Web服务器软件，也不用担心，本书将使用XAMPP作为Web服务器。

运行应用需要浏览器，推荐Firefox（同时安装Firebug插件）或Google Chrome。

在开始充满乐趣的实际操作之前，我们总结一下需要安装的工具。以下列出了下载地址，并且可找到安装说明。

- 带有调试工具的浏览器
  - Firefox with Firebug　https://www.mozilla.org/firefox/和 http://getfirebug.com/
  - Google Chrome　www.google.com/chrome
- Web服务器软件
  - XAMPP　http://www.apachefriends.org/en/xampp.html
- 数据库
  - MySQL　http://dev.mysql.com/downloads/mysql/
  - MySQL Workbench　http://dev.mysql.com/downloads/tools/workbench/
- MySQL Sakila样例数据库 http://dev.mysql.com/doc/index-other.html 和 http://dev.mysql.com/doc/sakila/en/index.html
- Sencha Command及其他所需工具
  - Sencha Command　http://www.sencha.com/products/sencha-cmd/download
  - Ruby　http://www.ruby-lang.org/en/downloads/
  - Sass　http://sass-lang.com/

- Compass　http://compass-style.org/
- Java JDK　http://www.oracle.com/technetwork/java/javase/downloads/index.html
- Java环境变量设置　http://docs.oracle.com/javase/tutorial/essential/environment/paths.html

当然还有Ext JS：http://www.sencha.com/products/extjs/，本书使用Ext JS 4.2版本。

1.2　展示应用及其功能

本书要开发的应用是个很常见的Web应用系统，你以前大概经常碰到。我们将实现一个视频商店管理程序（这也是我们使用Sakila样例数据库的原因），其典型功能包括安全管理（管理使用者及其权限），演员、影片、库存和租借信息管理等。

Ext JS将帮助我们实现目标，它提供了漂亮的组件，当用户看到一个由直观且友好的组件搭建而成的应用时，会感觉眼前一亮。对开发者而言，Ext JS提供了完整的解决方案，可以做到组件重用（减轻工作量），同时还有一套完整的数据包，使得与服务器端的通信以及信息的发送和获取得以简化。

我们把整个应用划分为若干模块，每个模块负责实现应用的某些功能。本书的每一章都将实现其中的一个模块。

应用的构成如下：

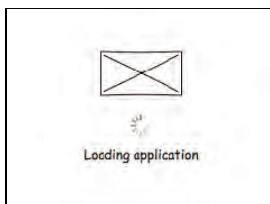
- 启动界面（应用启动时，用户看到的就不是空白界面了）；
- 登录界面；
- 主界面；
- 用户控制管理；
- MySQL数据库表管理（类别与组合框）；
- 内容管理控制；
- 电子邮件客户端模块。

对于上面提到的每个模块和界面，我们都将创建原型，以便规划应用如何工作。比如，是否应该使用菜单，点击菜单项之后出现的内容应该展示在窗体中、屏幕中央，还是标签面板里？

1.2.1　启动界面

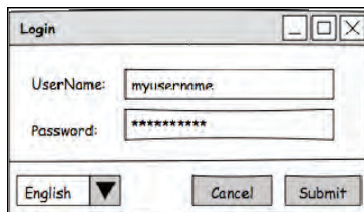
初次加载应用时，加载过程需要花费一些时间。如果我们不做点什么，用户就会看到一个空白页面，这显然非常无趣。

因此，应用应该有个启动界面，这样在应用初始化前加载所需文件或类时，用户就不用面对无趣的空白页面了。



1.2.2　登录界面

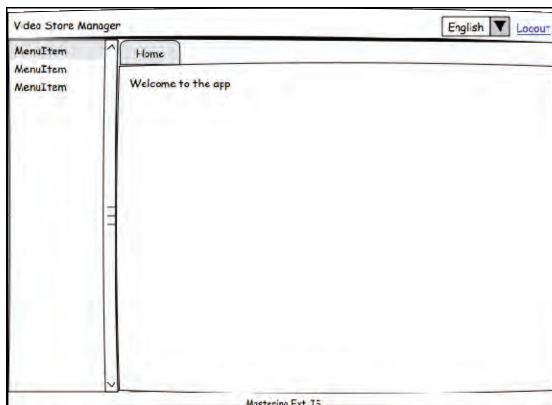
应用加载完成后，用户见到的第一个界面是Login（登录）界面。在这里，用户能够输入用户名（User Name）和密码（Password），同时还有个多语言组合框供用户选择系统语言。此外，界面中还有Cancel（取消）和Submit（提交）按钮。



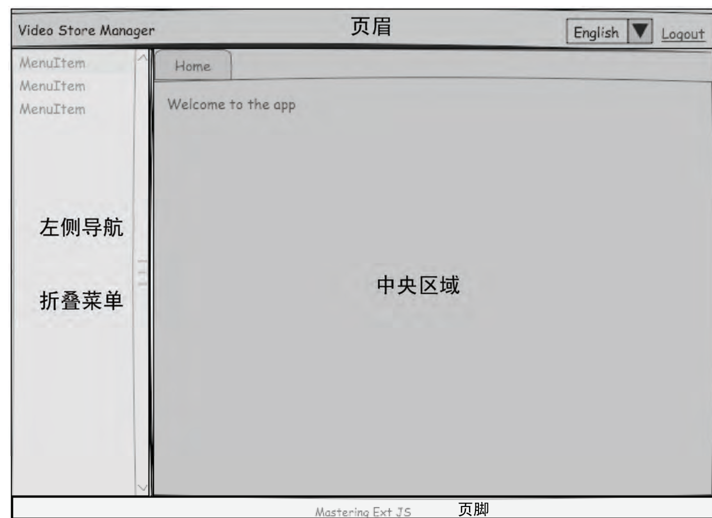
1.2.3　主界面

通常情况下，应用系统都会用边界布局（border layout）组织主界面。在中央区域，放置一个标签面板（tab panel），每个标签页表示应用的一个状态界面（每个界面有自己的布局），只有第一个标签页（主标签页，即Home页）不能关闭。界面顶部显示应用名Video Store Manager、多语言组合框和Logout按钮。界面底部包含版权信息（可以是公司名字或者项目开发者名字）。界面左侧有个动态菜单（用户管理），菜单用折叠面板（accordion panel）来实现（每个模块对应一个面板），每个面板中用树列出模块的菜单项。

主界面原型如下所示。



按照我们最初描述的布局分区，主界面的布局如下。



1.2.4　用户控制管理

在用户控制管理模块，使用者需创建新用户（New User）、新用户组（Groups），分配新角色给用户。用户可以控制系统权限（可以看到系统的各模块）。

1.2.5　MySQL数据库表管理

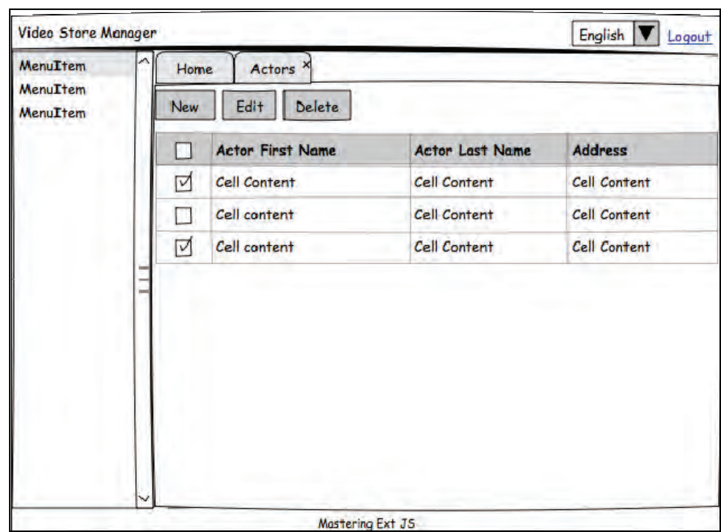
每个系统都有管理类别的选项Categories，比如影片类别、影片语言，以及组合框选项，等等。对于这些类别表，需要提供所有的CRUD选项和筛选选项，以下所示的模块界面与MySQL Workbench的Edit table data（编辑表数据）选项非常相似。

用户可以编辑表格行中的数据。

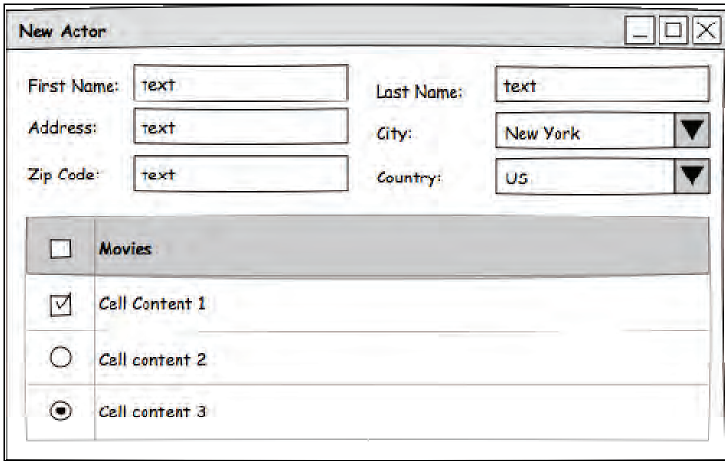
1.2.6　内容管理控制

在本模块，用户可以查看并编辑系统核心信息。在本模块中处理的大部分数据库表都与其他表关联，由于涉及主从关系，信息的编辑将变得复杂。一般情况下，网格面板用于呈现信息，而表单面板用于在打开的窗体中编辑信息。

模块的许多界面都有类似的功能，当我们要创建具有很多界面的应用时，要牢记一点：做好系统设计，尽可能重用代码，以便于维护、添加系统特性和功能。

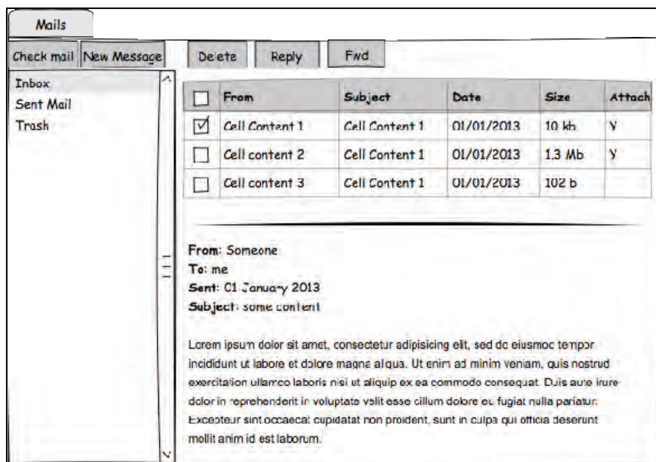


点击New（新建）或Edit（编辑）按钮，将打开一个编辑信息新窗体，如下图所示。



1.2.7　电子邮件客户端模块

在本模块中，我们将使用Ext JS设计一个电子邮件客户端。这很重要，因为这表明可以用Ext JS做很多事情，而不只是设计CRUD界面。本模块将实现电子邮件客户端的界面，但不实现发送和接受邮件的功能（这些功能要靠服务器端代码来实现）。



1.3　用 MVC 创建应用框架

让我们开始敲代码吧。第一件事就是用MVC架构创建应用。我们可以利用Sencha Command（Sencha Cmd）自动创建应用。Sencha Cmd对创建应用很有帮助，因为它根据MVC架构创建应用框架，并且提供创建产品应用以及定制主题所需的所有文件（后续章节会介绍具体做法）。

1.3.1　MVC简介

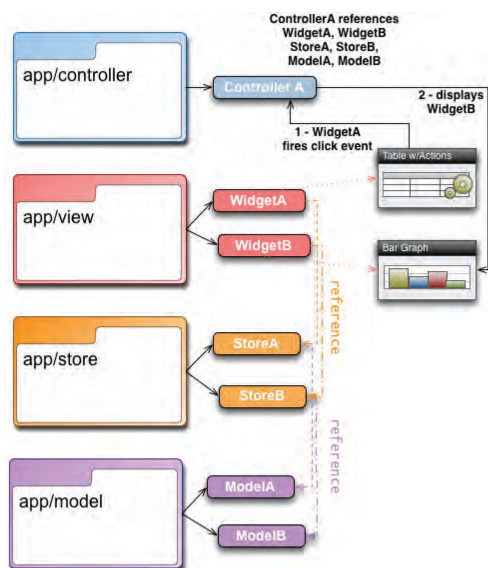
MVC是Model-View-Controller的缩写。它是一种软件架构模式，从用户交互信息中分离出呈现部分。Model（模型）表示应用数据，View（视图）表示数据输出展示（表单、表格、图表等），Controller（控制器）控制请求，将其转换为模型或视图的命令。

Ext JS使用MVCS，即Model-View-Controller-Store（模型–视图–控制器–存储器）模式。模型表示应用的数据，即数据库表。视图表示管理模型信息的所有组件和界面。Ext JS受事件驱动，当用户与之交互时，视图会触发事件，控制器捕捉这些事件，然后进行处理，重定向命令到模型（或存储器）或者视图。存储器在Ext JS中非常类似于服务器端常用的数据访问对象（Data Access Object，DAO）模式[①]。

举一个简单的例子。WidgetA是一个网格面板，用来显示数据库表A的所有记录，这个数据库表用ModelA来表示。用StoreA表示获取的信息（从服务器端获取的ModelA的集合）。当用户点击WidgetA上的一条记录时，将打开一个窗口（用WidgetB来表示）并显示数据库表B中的信息（用ModelB来表示）。显然，StoreB表示从服务器端获取的ModelB的集合。本例中，有一个ControllerA捕捉WidgetA发出的点击事件，并处理所有的请求逻辑，从而显示WidgetB并加载所有ModelB的

① DAO数据访问对象模式把数据访问操作和业务逻辑分开。——译者注

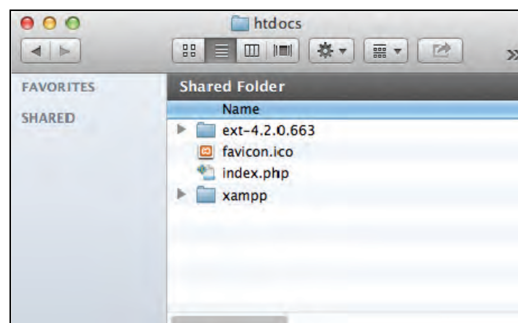
信息，参见下图。



1.3.2　创建应用

在xampp目录下的htdocs文件夹里创建应用，其名为masteringExtjs。

在继续之前，先看看htdocs文件夹。



里面有XAMPP文件和Ext JS 4.2文件。

下一步就是利用Sencha Cmd为我们创建应用。要运行Sencha Cmd，得先打开操作系统自带的终端工具，即Linux和Mac OS用户打开终端窗口，Windows用户打开命令行窗口。

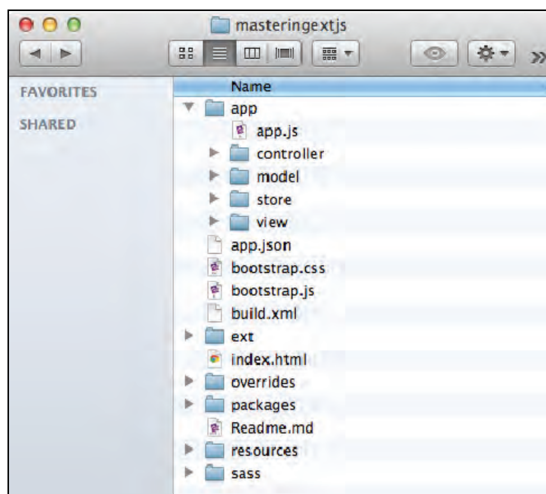
操作步骤如下：首先进入Ext JS目录（本例中为htdocs/ext-4.2.0.663），然后执行以下命令。

```
sencha generate app Packt ../masteringextjs
```

```
ext-4.2.0.663 — bash — 80×26
loiane:~ loiane$ cd /Applications/XAMPP/xamppfiles/htdocs/ext-4.2.0.663
loiane:ext-4.2.0.663 loiane$ sencha generate app Packt ../masteringextjs
Sencha Cmd v3.1.0.256
[INF]
[INF] init-plugin:
[INF]
[INF] init-plugin:
[INF] Invoking plugin (/Applications/XAMPP/xamppfiles/htdocs/ext-4.2.0.663/.senc
ha/workspace/plugin.xml) - supported targets: -before-generate-workspace
[INF]
[INF] -before-generate-workspace:
[INF] Invoking plugin (/Applications/XAMPP/xamppfiles/htdocs/ext-4.2.0.663/.senc
ha/workspace/plugin.xml) - supported targets: generate-workspace
[INF]
[INF] cmd-root-plugin.init-properties:
[INF]
[INF] init-properties:
[INF]
[INF] init-sencha-command:
[INF]
[INF] init:
[INF]
[INF] -before-generate-workspace:
[INF]
[INF] generate-workspace-impl:
[INF]      [echo] generating into /Applications/XAMPP/xamppfiles/htdocs/ext-4.2.
```

`sencha generate app`命令在htdocs文件夹中创建masteringextjs目录，并根据MVC架构所需创建文件结构。`Packt`是应用系统的命名空间，意味着创建的每个类都将以Packt打头，比如：`Packt.model.Actor`、`Pack.view.Login`，等等。传递给该命令的最后一个参数是应用程序目录，本例中，就是htdocs文件夹下的masteringextjs文件夹。

这条命令执行完毕，结果如下图所示：



为什么要创建这样一个项目结构呢？因为这就是Ext JS MVC应用要使用的结构。



若想了解更多关于sencha generate appcommand的信息，请访问http://docs.sencha.com/extjs/4.2.0/#!/guide/command_app。

下面，我们来看看每个文件夹的用途。

首先是app文件夹，我们写的所有应用代码都放在里面。app文件夹有以下子文件夹：controller、model、store和view，此外还有一个app.js文件[①]。接下来逐个介绍。

- 在model文件夹里创建模型文件，模型为拥有一组字段的Ext JS类，代表应用管理的对象（演员、地域和影片）。这非常类似于一个用来表示数据库表的服务器端的类，只有属性及对应的获取方法（getter）和设置方法（setter）。
- store文件夹存放存储器类，是模型集合的缓存。它们非常类似于*数据访问对象*（Data Access Object，DAO），即服务器端语言（如PHP）用于对数据库执行CRUD操作的类。因为Ext JS不具备跟数据库直接通信的能力，存储器类通过代理与服务器端或本地存储通信（存储器类用代理进行模型数据的加载和保存）。
- view文件夹存放视图类，即通常所说的UI组件（User Interface Components，*用户界面组件*），如网格面板、树形面板、Menu、表单面板、Window，等等。视图类只处理界面呈现，不处理组件的事件触发（Grid、Tree、Menu、Form、Window都是`Component`的子类）。
- 最后，controller文件夹存放控制器类，控制器类处理组件的事件触发（事件触发源于组件的生命周期，或者用户与组件的交互）。要牢记Ext JS是事件驱动的，我们在控制器类中控制事件，并在必要时修改模型、视图或存储器。

还有一个app.js文件，它是应用的入口，后续将用几段文字来描述它。

回到masteringextjs目录，里面包含几个文件和目录。

- **app.json**　Sencha Cmd的配置文件，打开它会发现里面只有一个与应用（`Packt`）同名的JSON对象。
- **bootstrap.css和bootstrap.js**　这两个文件都是Sencha Cmd创建的，不要修改它们。bootstrap.css存储应用系统使用主题的引用（蓝色经典主题）；bootstrap.js则存储一些请求指令，自定义的`xtype`属性以及一些元数据驱动类的系统特性。
- **build.xml**　Sencha Cmd使用Apache Ant（http://ant.apache.org/，一个Java项目的生成工具）。Ant使用的配置文件叫build.xml，包含了生成项目需要的所有配置和命令。Sencha Cmd使用Ant引擎在后台生成Ext JS应用（只需要一条简单的命令）。这也就是为什么需要安装Java SDK以使用Sencha Cmd的某些特性。
- **index.html**　这是项目的首页。运行应用时，浏览器将其呈现出来。在这个文件里，可以看到bootstrap CSS和JS文件的引用，以及Ext JS框架文件的引用（ext/ext-dev.js和app/app.js文件）。
- **ext**　这个文件夹存放所有的Ext JS框架文件（xt-all、ext-all-debug、ext-dev）及其源文件。

---

① 翻译本书时，Sencha Cmd最新版本3.1.2.342，app.js放在masteringextjs目录下。—— 译者注

- overrides　应用创建伊始，这个文件夹是空的。里面会创建一些Ext JS重写代码来满足我们的项目开发需要。
- packages　在这个文件夹里可以看到Sencha Cmd管理的所有包。关于包的更多信息，请访问：http://docs.sencha.com/extjs/4.2.0/#!/guide/command_packages。
- resources　这个文件夹放置创建应用所需的所有CSS样式文件（自定义样式、定位图标的CSS等），以及所有静态文件（图片）。
- sass　在这个文件夹里，可以看到用于创建主题的Sass文件。

下面，我们开始动手编码吧！

首先，我们需要编辑app.js文件，其原始代码如下：

```
Ext.application({
    name: 'Packt',

    views: [
        'Main',
        'Viewport'
    ],

    controllers: [
        'Main'
    ],

    autoCreateViewport: true
});
```

需要将其修改成下面这样：

```
Ext.application({ // #1

    name: 'Packt', // #2

    launch: function() { // #3
         console.log('launch'); // #4
    }
});
```

上面代码的第一行声明了一个Ext.application（#1），表示应用有一个页面，应用的父容器为Viewpoint。Viewpoint是一个特殊的容器，表示应用的可视区域，在HTML页面的body标签里渲染，决定应用在浏览器中的显示尺寸及窗体缩放。

在Ext.application中，还可以声明应用使用的模型、视图、存储器以及控制器。在后续创建项目新类时将不断把相关信息加进去。

我们需要声明应用名称，并作为命名空间（#2）。

我们还可以在Ext.application里创建一个启动（launch）函数（#3）。这个函数将在应用

的所有控制器初始化完成后被调用，这也意味着应用完成了加载。因此，这里是实例化主视图的合适位置。现在，只需加上console.log语句（#4），就可以在浏览器的JavaScript解释器控制台打印出信息，以此检验应用是否加载成功。

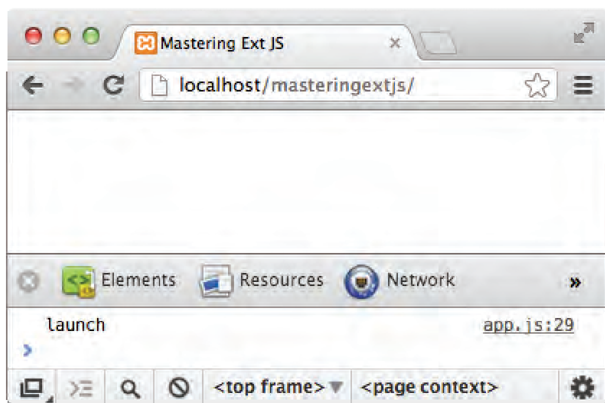


用了Ext.application，还需要用Ext.onReady吗？ 答案是不需要。二者只用一个即可。根据Ext JS API文档描述，当所有所需脚本完全加载，页面准备就绪且Ext.onReady添加新的监听器并执行之后，Ext.application才会加载Ext.app.Application类并用所给的配置将其启动。我们来看一下Ext.application的源代码：

```
Ext.application = function(config) {
    Ext.require('Ext.app.Application');
    Ext.onReady(function() {
        new Ext.app.Application(config);
    });
};
```

这说明Ext.application已经调用了Ext.onReady，所以无须再次调用。当只有少量组件要显示，且未使用MVC架构时，可以使用Ext.onReady（类似于jQuery的$(document).ready()函数）；当开发一个Ext JS MVC应用程序时，可以使用Ext.application。

通过访问http://localhost/masteringextjs，可以在浏览器运行应用程序，结果如下图所示。

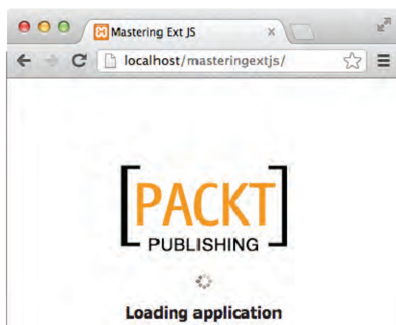


接下来，我们准备创建应用。

1.4　创建加载页面

大型的Ext JS应用系统通常在加载时会有短暂的延时，这是因为Ext JS需要加载所有所需的类以保障应用的启动和运行，这意味着此时用户会看到一个空白页面，这有点不讨人喜欢。解决这个问题的通用方案就是添加一个加载页面，也就是常说的启动界面。

所以，需要为应用添加一个如下图所示的启动界面。



首先，我们需要理解启动界面的工作原理。当用户加载应用时，加载页面呈现出来。当应用加载全部所需的类以及代码时，则呈现启动界面。

我们已经知道当应用准备就绪可供使用时，会调用启动函数。所以，启动界面的实现逻辑不能放在启动函数中。那现在的问题就是：具体在`Ext.application`的何处可以调用启动界面的实现逻辑呢？答案就是在`init`函数里。`init`函数在应用程序启动时被调用，所以给所需代码的加载腾出了一定的时间，之后启动函数才被调用。

现在我们了解了启动界面的工作原理，下一步就是实现它。

在`Ext.application`中实现`init`函数：

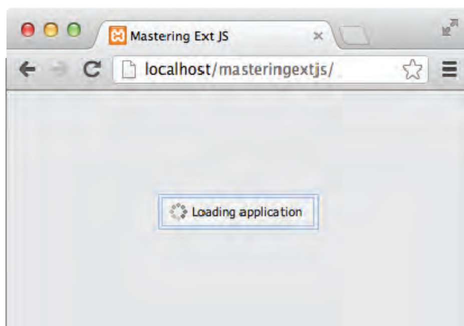
```
init: function() {
     splashscreen = Ext.getBody().mask('Loading application',
     'splashscreen');
},
```

我们要做的就是在HTML body标签上（`Ext.getBody()`）插入一个遮罩，因此要调用`mask`方法，传递加载信息（Loading Application），并应用CSS样式。后面，还将加载一个gif动画（也是Ext JS CSS样式splashscreen里的一部分）。`mask`方法返回`Ext.dom.Element`对象，后续还会用到它（移除遮罩）。因此，我们需要保留一个`Ext.dom.Element`的引用，并把这个引用保存在

Ext.application的一个属性里：

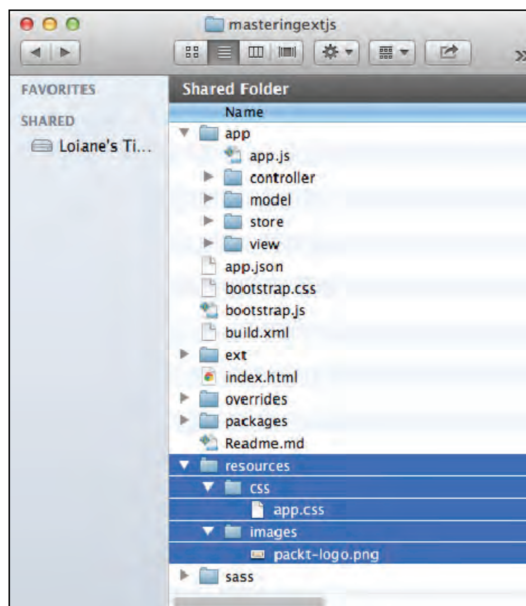
```
splashscreen: {},
```

现有init方法的代码所呈现的加载页面如下图所示：



如果这是你需要的界面，那这样就足够了。但让我们再更进一步，通过加上标志，让它看起来像本节开头的图示那样，这才是最终像样的画面。

首先，在resources文件夹下的css子文件夹中创建一个CSS样式表，包含应用的所有样式设置，并将其命名为app.css：



在resources文件夹中再创建一个images文件夹，里面放Packt的logo图片。

别忘了在index.html里加入新的CSS文件引用。

```
<link rel="stylesheet" href="resources/css/app.css">
```

app.css文件包含如下代码：

```
.x-mask.splashscreen {
    background-color: white;
    opacity: 1;
}

.x-mask-msg.splashscreen,
.x-mask-msg.splashscreen div {
    font-size: 16px;
    font-weight: bold;
    padding: 30px 5px 5px 5px;
    border: none;
    background-color: transparent;
    background-position: top center;
}

.x-message-box .x-window-body .x-box-inner {
    min-height: 110px !important;
}

.x-splash-icon {
    background-image: url('../images/packt-logo.png') !important;
    margin-top: -30px;
    margin-bottom: 15px;
    height: 100px;
}
```

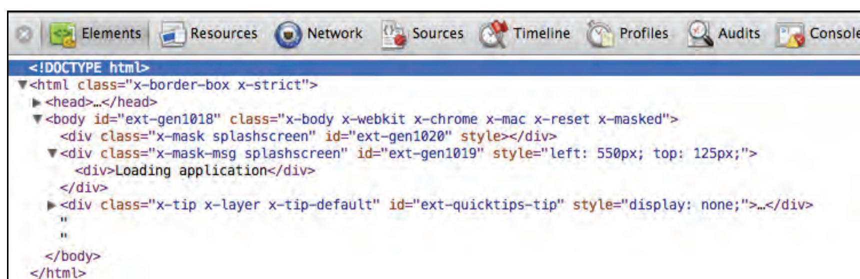
现在回过头来看app.js文件，继续在init函数里添加代码。

如果在init函数现有代码的后面添加以下代码：

```
splashscreen.addCls('splashscreen');
```

将添加新的CSS样式用以加载DIV标签。注意，此时应用的是app.css文件里的.x-mask.splashscreen和.x-mask-msg.splashscreen div样式。这将导致背景由灰色变为白色，同时，“Loading Application”的字体也将改变。

生成的HTML代码如下：

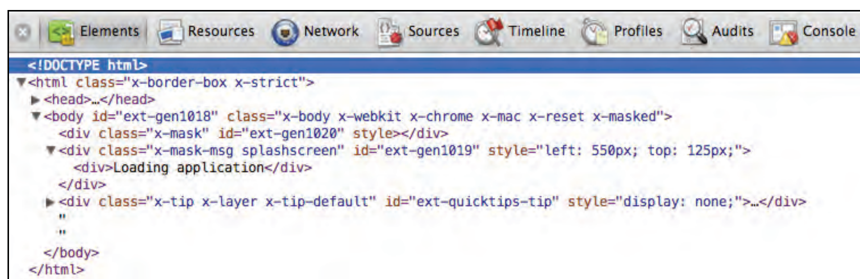


现在，在`init`函数里加入以下代码：

```
Ext.DomHelper.insertFirst(Ext.query('.x-mask-msg')[0], {
     cls: 'x-splash-icon'
});
```

此行代码将查找第一个包含`.x-mask-msg`类（`Ext.query('.x-mask-msg')[0]`）的`DIV`标签，然后添加一个新的`DIV`标签作为子标签，样式类为`x-splash-icon`，负责在加载信息上方添加logo图片。

生成的HTML代码如下：



执行上述代码后，将得到本节开头所展示的结果。

现在已经实现了启动界面。接下来，需要在启动函数中移除启动界面，否则，加载信息将一直显示。

移除启动界面的代码只有一行：

```
Ext.getBody().unmask();
```

但是，遮罩消失得太突然并不好，因为用户甚至有可能看不到加载信息。更好的方案是在应用就绪后留2秒钟给用户，以便其可以看到加载信息。

```
var task = new Ext.util.DelayedTask(function() { // #1
     Ext.getBody().unmask(); // #2
});

task.delay(2000); // #3
```

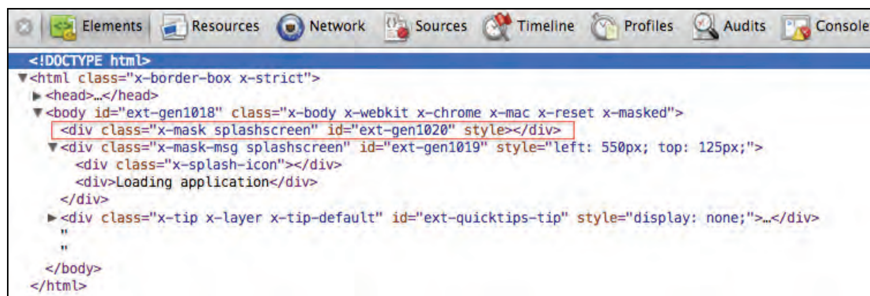
如此一来，我们就要用到DelayedTask类（#1），它能使函数在过了设定时间（#3，以毫秒为单位）后才执行。在本例中，两秒（2000毫秒）过后遮罩才会消失（#2）。

如果现在查看输出结果，会发现程序能够正常运行，但对用户而言仍不够友好，如果在遮罩中加个动画会更好。现在，我们给其加上一个淡出动画（通过动画形式，把元素的透明度逐渐提高，使其由不透明变为透明）。动画过后，仍要移除这个遮罩（在Ext.util.DelayedTask函数中）。

```
splashscreen.fadeOut({
    duration: 1000,
    remove:true
});
```

执行完这段代码后，我们发现加载信息仍然显示着。需要分析HTML代码找出原因。

执行fadeOut函数之前，加载信息的HTML代码如下图所示：



执行完fadeOut函数后，HTML代码如下图所示：

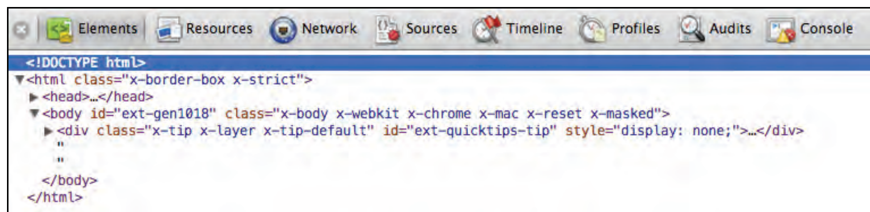


只有第一个具备splashscreen样式类的DIV标签产生了淡出效果。还需要具备x-mask-msg splashscreen样式、包含logo标识和加载信息的DIV标签也产生淡出效果。

```
splashscreen.next().fadeOut({
    duration: 1000,
    remove:true
});
```

这样一来，退出动画看起来就舒服了。请注意，此时具备splashscreen样式的DIV标签已

经从HTML代码中移除了：



遮罩被移除后，需要呈现应用的组件界面。接下来要呈现一个将在下一章实现的登录界面。现在，我们先加个控制台信息（#1），标注一下需要在什么位置调用组件处理逻辑，完成的启动函数的代码如下所示：

```
launch: function() {
    var task = new Ext.util.DelayedTask(function() {

        splashscreen.fadeOut({
            duration: 1000,
            remove:true
        });

        splashscreen.next().fadeOut({
            duration: 1000,
            remove:true
        });

        console.log('launch'); // #1
    });

    task.delay(2000);
}
```



用于显示及移除遮罩的代码（方法）都是Ext.dom.Element类的一部分。这个类封装了文档对象模型（Document Object Model，DOM）元素，可以用类方法来管理这些元素。这个类属于Ext JS核心库，核心库是Ext JS框架的基础。

1.5　小结

本章介绍了本书各章要致力于实现的应用的基本情况，以及搭建应用开发环境所需的工具，并且学习了怎样创建一个基于Ext JS MVC框架的应用，其初始结构如何。

此外，通过实例，我们还掌握了创建启动界面（也叫加载界面）的方法，即用Ext.dom.Element类管理DOM。我们了解了Ext.onReady与Ext.application的区别，以及Ext.application的init与launch方法的不同。下一章，我们将进一步完善app.js文件以显示应用的第一个界面：登录界面。

第 2 章

登录界面

2

应用系统通常都有一个登录界面，可以通过用户提供的认证信息鉴别、验证用户，从而达到控制系统访问的目的。一旦用户登录，系统就可以跟踪用户的行为。我们还可以设置应用系统的一些特性和界面的访问权限，限制特定用户或某一用户组的访问。

在本章中，我们将介绍以下内容：

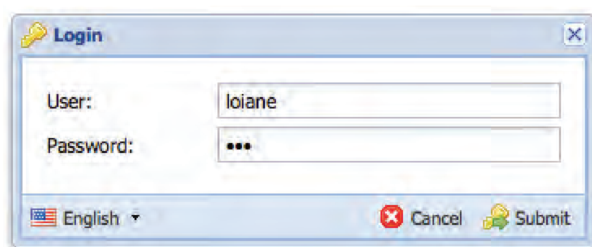
- 创建登录界面；
- 在服务器端处理登录界面；
- 在Password（密码）字段添加大写键提醒；
- 按回车键提交表单的功能；
- 在信息发送给服务器端之前加密密码。

2.1 登录界面

Login窗口是本项目要实现的第一个视图。我们将一步步创建它，使其最终具备以下功能：

- 用户通过键入用户名和密码登录系统；
- 客户端验证（登录时必须提供用户名和密码）；
- 用户按回车键提交Login表单；
- 在信息发送给服务器端之前加密密码；
- 多语言支持。

除了将要在下一章实现的多语言支持功能，其余功能都将在本章完成。创建完成后，Login窗口将如下图所示：



让我们开始吧！

2.2　创建登录界面

在app/view目录下，创建一个名为Login.js的新文件。这个文件将包含在屏幕上创建可见元素的所有代码。

在Login.js文件里，写出如下代码：

```
Ext.define('Packt.view.Login', { // #1
    extend: 'Ext.window.Window', // #2
    alias: 'widget.login',       // #3

    autoShow: true,              // #4
    height: 170,                 // #5
    width: 360,                  // #6
    layout: {
        type: 'fit'              // #7
    },
    iconCls: 'key',              // #8
    title: "Login",              // #9
    closeAction: 'hide',         // #10
    closable: false              // #11
});
```

第一行（#1）用`Ext.define`定义了一个类，其后紧跟一对圆括号（`()`），在圆括号内首先声明类名，后面是逗号（,）以及花括号（`{}`），最后是一个分号。所有的配置参数和属性（#2~#11）都放在花括号内。

请注意类名，Sencha对Ext JS MVC项目的建议命名规则是：应用的命名空间+包名+JavaScript文件名。上一章，我们定义了命名空间Packt（在app.js文件里）。同时，我们还创建了项目的视图结构，因此创建的JavaScript文件应该放在view包（目录）中。这里的JavaScript文件名是Login.js，去掉.js，Login就是视图名称。于是，我们的类名就是`Packt.view.Login`。

`Login`类继承`Window`类（#2），因为我们希望登录界面在一个窗体内呈现。

我们还分配给了`Login`类一个别名`alias`（#3）。从组件扩展而来的类的别名总是以`widget.`

打头，后面跟着我们取的别名。别名的命名惯例是小写。还有一点很重要：别名在应用中必须是唯一的。本例中以login作为这个类的别名，后面就可以用别名实例化这个类了（跟使用xtype一样）。举个例子，可以用四种不同的方式实例化Login类。

- 用类的全称，这是最常用的方式：

```
Ext.create('Packt.view.Login');
```

- 在Ext.create方法中使用别名：

```
Ext.create('widget.login');
```

- 在Ext.ClassManager.instantiateByAlias的快捷方式Ext.widget中使用别名：

```
Ext.widget('login');
```

- 使用xtype作为另一组件的项：

```
items: [
  {
    xtype: 'login'
  }
]
```

本书大多数情况下用第一种、第三种和第四种方式。

接下来，设置autoShow（自动显示）为true（#4），对于窗体而言，仅仅实例化组件是不足以呈现它的。实例化窗体后，我们拥有了窗体的引用，但它还不能在屏幕上呈现出来。如果想让它呈现出来，就还需要手动调用show()方法。另一个让它呈现出来的方式是将autoShow属性设置为true。这样，当窗体实例化后就会自动呈现出来了。

同时，我们还需设置窗体的height（#5，高度）和width（#6，宽度）。设置布局类型为fit（#7），因为我们希望登录表单（包含用户名和密码字段）占据整个窗体。但要注意，用了fit布局就可以只设置一个子组件项了。

设置窗体的iconCls（#8）属性，就可以为窗体的标题栏添加“钥匙”图标。还可以设置窗体的标题，这里为“Login”（#9）。下面这行代码声明了iconCls属性用到的“钥匙”图标样式：

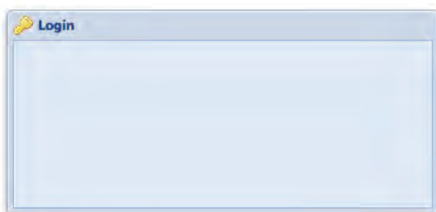
```
.key {
  background-image:url('../icons/key.png') !important;
}
```

我们为iconCls创建的所有样式都要像上面这样来设置。

最后，我们设置了closeAction（#10） 和closable（#11）两个属性。closeAction设置关闭窗体时是否销毁窗体。本例中，我们并不想销毁窗体，而只想隐藏它。closable设置在窗体右上角是否呈现关闭图标（一个叉），因为这是Login窗体，所以没必要呈现。

如果你愿意，还可以加上draggable和resizable属性并设置为false，用于阻止用户拖曳、缩放Login窗体。

截至目前，我们完成了一个左上角有标题Login及“钥匙”图标的窗体：



下一步，我们添加一个带有用户名和密码字段的表单。把以下代码添加到Login类中：

```
items: [
    {
        xtype: 'form',          // #12
        frame: false,           // #13
        bodyPadding: 15,        // #14
        defaults: {               // #15
            xtype: 'textfield', // #16
            anchor: '100%',     // #17
            labelWidth: 60      // #18
        },
        items: [
            {
                name: 'user',
                fieldLabel: "User"
            },
            {
                inputType: 'password', // #19
                name: 'password',
                fieldLabel: "Password"
            }
        ]
    }
]
```

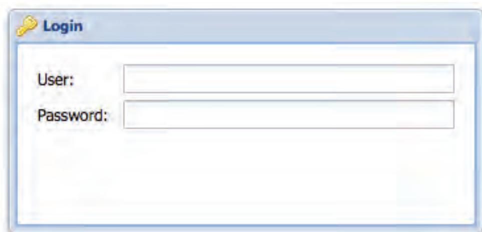
前面用了fit窗体布局，那就只需要在Login类中声明一个子项（item）。因此，我们添加一个表单（#12）并打算让这个表单看起来更漂亮些：移除frame属性（#13），设置表单的填充属性（#14）。表单的frame属性默认值就是false，但如果我们不明确将属性设置为false的话，还是会有个蓝色边框出现。

我们要在表单上添加两个字段，同时还要避免代码重复。这也是为什么在表单的defaults配置项里设置一些属性的原因（#15）。这里的设置会影响表单中的所有元素，我们在其中只设置想定制的内容。我们想在表单中声明两个字段，都是textfield类型，而表单默认的布局是anchor，因此不需要明确声明它。但是，我们希望这两个字段在水平方向占据表单内的所有空

间，所以声明anchor属性为100%（#17）。默认情况下，TextField标签（label）的宽度（width）为100像素，这对用户名和密码字段来说太长了，我们把它缩小为60像素（#18）。

最后，我们设置了user和password两个文本字段，name属性用来在提交表单到服务器端时识别相应字段。还剩下一个问题：当用户在密码框里输入密码时，系统不能显示密码明文。所以，我们将password字段的inputType属性设置为'password'（#19）。这样，用户输入的密码就会显示为圆点，而不是明文了。

我们又进一步完善了Login窗体，目前效果如下图所示：



2.2.1　客户端验证

Ext JS的表单字段组件具有一定的客户端验证能力，这可以节省我们的时间和带宽（系统只有在通过基本验证之后才会发起服务器端请求）。此外，客户端验证还能指出用户填表单时哪儿出了错。当然，出于安全原因，仍需要服务器端验证，但就现在而言，我们主要关注Login表单的客户端验证。

思考一下用户名和密码字段的验证场景。

- 用户名和密码是必填项，但怎样验证没有用户名和密码的用户呢?
- 在两个字段中，用户只能输入字母和数字（A~Z、a~z以及0~9）。
- 用户名字段只能输入3~25个字符。
- 密码字段也只能输入3~15 个字符。

因此，加入对两个字段都通用的代码：

```
allowBlank: false, // #20
vtype: 'alphanum', // #21
minLength: 3,      // #22
msgTarget: 'under' // #23
```

我们将把上述配置加到表单的defaults属性配置中，这些配置将同时作用于用户名和密码字段。首先，两个字段都是必填项（#20），只允许输入字母和数字（#21），用户至少要输入3个字符（#22）。然后是最后一个通用配置：在字段下面显示验证错误信息（#23）。

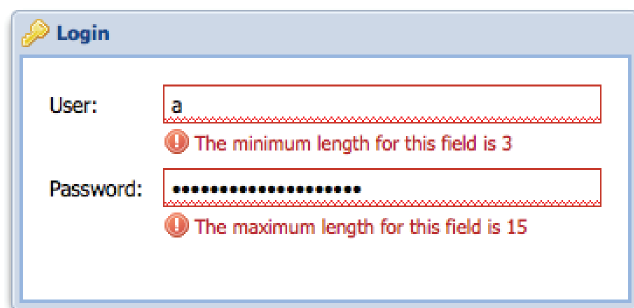
需要自定义的是User（用户名）字段最大字符数为25：

```
name: 'user',
fieldLabel: "User",
maxLength: 25
```

Password（密码）字段最大字符数为15：

```
inputType: 'password',
name: 'password',
fieldLabel: "Password",
maxLength: 15
```

下图展示了应用客户端验证之后，用户填写Login窗体出错时的场景：



如果你不喜欢这种样式，我们可以改变出错信息的显示位置，只需改变msgTarget属性值即可。msgTarget属性选项有：title、under、side和none。此外还能以工具提示（qtip）风格显示出错信息，或者在特定目标（的内部HTML）上显示。

创建自定义的VType

许多应用系统都具有特殊的密码格式。不妨举个例子：我们的密码需要至少一个数字（0~9）、一个小写字母、一个大写字母、一个特别字符（@、#、$、%……），长度在6~20个字符之间。那么，可以通过正则表达式来验证输入的密码，并创建一个自定义的VType来实现验证。创建自定义的VType很容易，在本例中，我们创建一个名为customPass的自定义VType：

```
Ext.apply(Ext.form.field.VTypes, {
    customPass: function(val, field) {
        return /^((?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%]).{6,20})/.test(val);
    },
    customPassText: '不是一个合法密码。密码长度应在6~20个字符之间，必须包含至少一个数字、一个
小写字母、一个大写字母、一个特别字符（@#$%）.',
});
```

customPass是自定义VType的名称，同时需要声明一个验证正则表达式的函数。customPassText是当输入不正确密码格式时显示的出错信息。

上面代码可以放到任意地方：控制器的`init`函数、`app.js`的启动函数，或者是独立存放所有自定义VType的JavaScript文件里（推荐）。

实际使用时，只需简单地添加`customPass` VType到我们的密码字段即可。



若希望了解更多关于正则表达式的内容，请参考http://www.regular-expressions.info/。

2.2.2　添加带有按钮的工具栏

截至目前，我们已经创建了Login窗体，在其中添加了带有两个字段的表单，并且完成了验证。剩下的一项任务就是添加cancel（取消）和submit（提交）按钮。

我们准备添加一个`toolbar`（工具栏），将取消和提交按钮作为其子项（item），工具栏作为表单停驻项（docked item）。停驻项可以停驻在面板的任意位置。本例中，工具栏停驻在表单底部。在表单items属性配置的后面添加以下代码：

```
dockedItems: [
    {
        xtype: 'toolbar',
        dock: 'bottom',
        items: [
            {
                xtype: 'tbfill' //#24
            },
            {
                xtype: 'button', // #25
                itemId: 'cancel',
                iconCls: 'cancel',
                text: 'Cancel'
            },
            {
                xtype: 'button', // #26
                itemId: 'submit',
                formBind: true, // #27
                iconCls: 'key-go',
                text: "Submit"
            }
        ]
    }
]
```

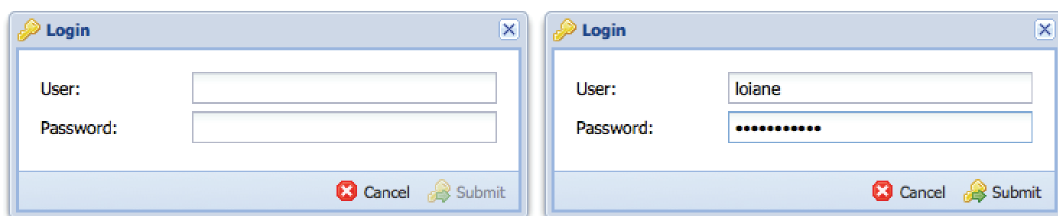
回顾本章开头第一次呈现的Login界面，我们会注意到有一个具有多语言转换功能的组件。这个组件后面是一块空白，然后是Cancel和Submit按钮。因为现在还没实现多语言组件，我们就先只实现两个按钮，但按钮需要出现在表单右端，同时还需要预留空白。这就是我们为什么首先要在`toolbar`上添加一个`fill`组件（#24）的原因——实现工具栏按钮右对齐。

接下来依次添加Cancel按钮（#25）和Submit按钮（#26）。再给两个按钮添加上图标（iconCls），后续还要实现控制类，对按钮进行识别，所以，我们给这两个按钮分配了itemId。

现在我们已经有了客户端验证，但即使有了这种验证，用户仍可以点击Submit按钮提交表单，我们应该避免这种情况的发生。因此，把Submit按钮绑定到表单（#27），这样，只有表单的客户端验证通过时，才能启用提交按钮。

2

下面是Login表单（已经添加了工具栏）在客户端验证通过情况下的输出图示：



2.2.3 运行代码

要运行目前已完成的代码，我们还需要对app.js做一些调整。

首先，声明我们要用到的视图（本例中只有一个）。此外，当用Login类的xtype属性进行实例化时，还需要在requires中声明这个类：

```
requires: [
    'Packt.view.Login'
],

views: [
    'Login'
],
```

最后需修改启动函数。上一章我们在需要实例化视图的地方用console.log信息占位，现在就用Login实例化代码来取代它（#1）。

```
splashscreen.next().fadeOut({
    duration: 1000,
    remove:true,
    listeners: {
        afteranimate: function(el, startTime, eOpts ){
            Ext.widget('login'); // #1
        }
    }
});
```

app.js调整完毕，可以运行完成的代码了！

2.2.4　**itemId**还是**id**：Ext.Cmp的问题

在创建控制器之前，还需要掌握一些关于Ext.ComponentQuery选择器的知识。本节讨论的话题将帮助我们更好地理解创建Login窗体及后续实现控制器的思路。

只要有可能，我们就使用itemId属性而非id属性来唯一标识某个组件。为什么呢？

使用id时，要确保id的唯一性，应用的其他组件不能使用同一id。想像一下：你跟其他开发者在一个项目团队里共同负责一个大应用系统，怎么确保id唯一呢？相当困难，几乎做不到。

可以通过Ext.getCmp全局访问带有id的组件，Ext.getCmp是Ext.ComponentManager.get的快捷方式。

举个例子，当使用Ext.Cmp通过id获取组件时，将返回最后使用此id的那个组件，如果id不唯一，返回的组件可能不是你想要的，同时也会导致应用错误。

别惊慌，我们有个很好的解决方案，那就是用itemId替代id。

itemId可作为获取组件引用的替代方式。itemId是容器内部的混合集合（MixedCollection）的索引，所以itemId身处容器的本地作用域，这是itemId的最大优势。

例如，有个名为MyWindow1的类，扩展自window，在这个类里，有一个itemId为'submit'的按钮。还可以有另一个名为MyWindow2的类，也扩展自window，同样有一个itemId为'submit'的按钮。

两个itemId的值相同并不是什么问题，我们只需关心如何通过Ext.ComponentQuery获取我们需要的组件。例如，有一个别名为login的Login窗体，另有一个别名为registration的Registration（注册）窗体，两个窗体都有Save（保存）按钮，其itemId都为'save'，如果只是使用Ext.ComponentQuery.query('button#save')，将返回一个包含两个结果的数组。然而，如果进一步缩小选择器的范围，假设我们只想获取Login窗体而非Registration窗体的Save按钮，就需要使用Ext.ComponentQuery.query('login button#save')，结果就是Login窗体的Save按钮，正是我们所需的。

注意，我们在此并未使用Ext.getCmp，因为使用它不是最佳实践，尤其不适合Ext JS 4。可以使用itemId和Ext.ComponentQuery替代它。下一节我们将更进一步地理解Ext.ComponentQuery。

2.3　创建登录控制器

现在我们已经创建了Login界面的视图。遵循MVC架构模式，我们还需要实现基于视图类的用户交互。这时点击Login窗体的按钮是没有任何反应的，因为尚缺处理逻辑。下面我们就来实现控制器类的处理逻辑。

在app/controller目录下，创建一个名为Login.js的新文件。在此文件里将实现所有与Login窗体事件管理相关的代码。

Login.js文件里，先实现以下代码，这还只是controller（控制器）类的基础代码：

```
Ext.define('Packt.controller.Login', { // #1
    extend: 'Ext.app.Controller',       // #2

    views: [
        'Login'                         // #3
    ],

    init: function(application) {       // #4
        this.control({                  // #5

        });
    }
});
```

通常，类的第一行是它的名称（#1）。遵循与view/Login.js同样的命名原则，Packt（应用的命名空间）+controller（包名）+Login（文件名），所以命名结果应该是Packt.controller.Login。



控制器JS文件controller/Login.js和view/Login.js文件同名。由于它们在不同的包里，所以这没什么问题。视图、模型、存储器和控制器具有相同的名称，这有利于后续更方便地管理项目。比如，项目完成上线时需要在登录窗体添加一个新的按钮。只根据这点信息（以及少许的MVC概念知识）我们就知道要在view/Login.js里添加按钮代码，要在controller/Login.js里实现按钮监听事件代码。易维护性是MVC架构的优势所在。

控制器类需要扩展Ext.app.Controller（#2），因此我们总是要把它作为控制器类的父类。

接下来是视图声明（#3），控制器关联的视图都在这里声明，目前只有登录（Login）视图，后面还会添加更多的视图。

下一步，声明init方法（#4）。init方法在应用引导启动前，先于Ext.application（app.js）的launch方法被调用。控制器也会加载其类中声明的视图、模型和存储器。

然后，设置control方法（#5），监听控制器需作出反应的所有事件。此外，处理Login窗体及其组件事件的代码也写在这里。

2.3.1 在app.js中添加控制器

现在有了基本的登录控制器代码，还需要在app.js中添加控制器。

移除以下代码，因为控制器将负责加载view/Login.js操作。

```
requires: [
    'Packt.view.Login'
],
views: [
    'Login'
],
```

然后加上控制器声明：

```
controllers: [
    'Login'
],
```

项目伊始就在控制器类里声明视图将有助于我们组织代码，这样就不需要在app.js里声明所有应用的视图了。

2.3.2 监听按钮点击事件

下一步开始监听Login窗体的事件。首先，监听Submit和Cancel按钮。

我们已经了解到需要在`this.control`声明里添加监听器，下面是相应处理方式：

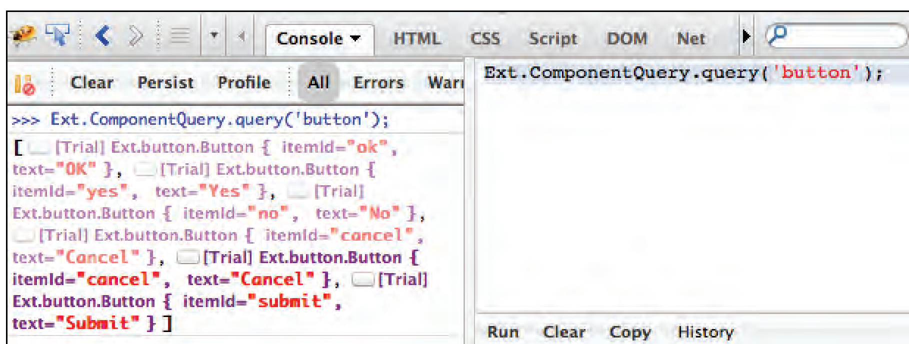
```
'Ext.ComponentQuery selector': {
  eventWeWantToListenTo: functionOrMethodWeWantToExecute
}
```

首先，传入`Ext.ComponentQuery`类要用到的选择器以找到组件，然后列出希望监听的事件。接下来，声明监听事件触发的执行函数，或声明事件触发时执行的控制器方法的名称。在本例中，我们声明的方法只是为了保持代码结构的完整性。

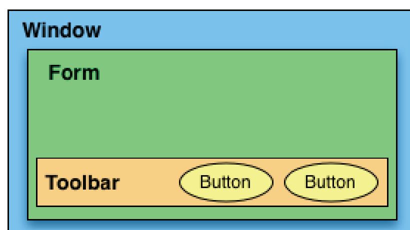
现在，我们将重点放在如何准确找到Submit和Cancel按钮的选择器上。根据`Ext.ComponentQuery` API文档的描述，可以通过`xtype`取回组件。（如果你熟悉jQuery，就会发现`Ext.ComponentQuery`选择器与jQuery选择器的行为非常类似。）下面我们要取回这两个按钮，它们的`xtype`是`button`。注意在编码之前，需验证一下选择器是否正确，才能避免多次调整代码。有个小技巧：打开浏览器控制台，输入以下命令，然后点击Run（运行）：

```
Ext.ComponentQuery.query('button');
```

从截图中可以看到，用`'button'`选择器查找返回了一个数组，里面竟然有6个按钮，这实在是太多了，不是我们期待的结果，我们只想要提交和取消按钮。



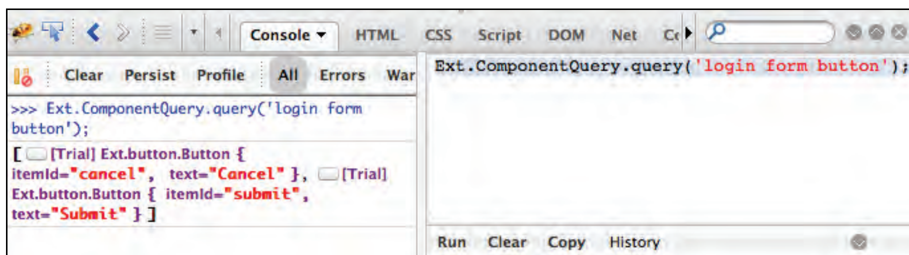
下面来使用Login窗体上组件的xtype画出其层次结构：



我们有一个Login窗体（xtype：login或window），窗体内有一个表单（xtype：form），表单内有一个工具栏（xtype：toolbar），工具栏内有两个按钮（xtype：button）。因此，可以得到选择器：login-form-toolbar-button。但是，如果我们用login-form-button，结果相同，这是因为表单内并没有其他按钮，来试试：

```
Ext.ComponentQuery.query('login form button');
```

在命令编辑器里试一下上述选择器：

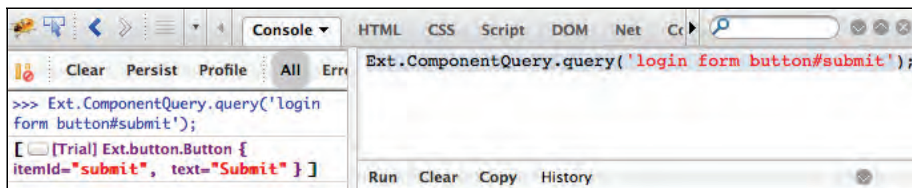


现在返回了仅包含两个按钮的数组！还有一个问题：如果使用login form button选择器，将监听这两个按钮的点击事件（虽然也是我们想监听的事件）。但是，我们知道：点击Cancel按钮将重置表单，而点击Submit按钮将提交表单数据至服务器端验证登录，这两个事件的处理方式不同，应该区别对待。因此，我们还想进一步缩小选择器范围，让一个选择器返回Cancel按钮，另一个选择器返回Submit按钮。

回顾一下view/Login的代码，我们给这两个按钮声明了itemId。可以采用通过itemId属性配置来识别按钮的独特方式，根据Ext.ComponentQuery API文档说明，可以使用“#”作为itemId的前缀。在命令编辑器里运行以下代码，获取Submit按钮的引用：

```
Ext.ComponentQuery.query('login form button#submit');
```

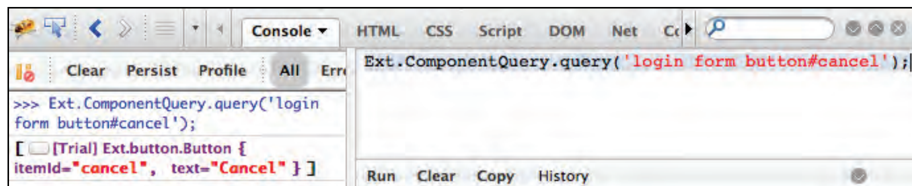
输出结果就是我们期待的唯一按钮（Submit按钮）：



再试一下获取Cancel按钮的引用：

```
Ext.ComponentQuery.query('login form button#cancel');
```

输出结果就是Cancel按钮：



终于找到合适的选择器了！控制台命令编辑器是个好工具，能帮我们节省大量时间，否则我们会一直编码、测试……

能否用button#submit或者button#cancel作选择器呢？是的，我们可以使用一个更短的选择器。但是，如此一来虽然现阶段能够正确工作，而当应用功能不断丰富，声明的类和按钮越来越多时，所有itemId名称为submit或cancel的按钮都将触发事件，应用就会报错。一定要牢记itemId是在容器的本地作用域里，用login form button作为选择器，就能确保事件来自Login窗体表单里的按钮。

因此，在控制器类里实现以下代码：

```
init: function(application) {
    this.control({
        "login form button#submit": {        // #1
            click: this.onButtonClickSubmit // #2
        },
        "login form button#cancel": {        // #3
            click: this.onButtonClickCancel // #4
```

```
            }
        });
    },

    onButtonClickSubmit: function(button, e, options) {
        console.log('login submit');              // #5
    },

    onButtonClickCancel: function(button, e, options) {
        console.log('login cancel');              // #6
    }
```

这里，先添加了一个Submit按钮的监听器（#1），接下来的一行说明我们想监听提交按钮的点击（`click`）事件，当Submit按钮的点击事件触发后，`onButtonClickSubmit`方法将被执行（#2，事件处理方法）。

之后，取消按钮也一样：声明一个Cancel按钮的监听器（#3），接下来一行监听Cancel按钮的点击事件，然后当Cancel按钮的点击事件触发时，执行事件处理`onButtonClickCancel`方法（#4）。

接下来，声明`onButtonClickSubmit`和`onButtonClickCancel`方法。当前，我们只在控制台输出一条信息，以确认代码可以运行。因此，在用户点击Submit按钮时输出`login submit`（#5），点击Cancel按钮时输出`login cancel`（#6）。

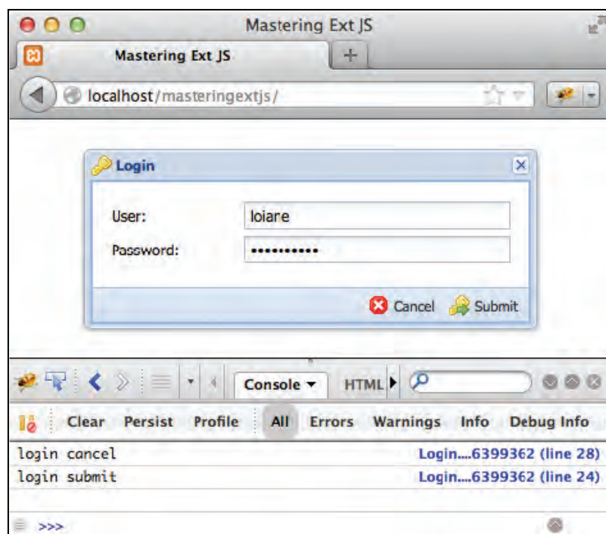
但如何知道事件处理方法可以接受哪些参数呢？去文档里找答案吧。看一下文档里关于点击事件的描述，可以看到：

```
click( Ext.button.Button this, Event e, Object eOpts )
Fires when this button is clicked, before the configured handler is invoked. ...
```

我们就是这么声明的。对于其他事件监听器，都去查看下文档，看看事件接受什么参数，然后在我们的代码里列出这些参数，这也是一个很好的实践。我们应该列出文档里描述的所有参数，即使只对第一个参数感兴趣。这种方式下我们就能始终掌握已有的完整参数集，便于维护应用系统。

让我们继续并试着运行一下。点击Cancel按钮，然后再点击Submit按钮。

运行结果如下所示：



1. 实现Cancel按钮点击事件处理方法

接下来移除`console.log`信息，并添加实际需要的代码。第一步，处理`onButtonClickCancel`方法。

编码逻辑顺序如下：

(1) 获取Login表单引用；
(2) 调用`getForm`方法，返回一个表单基类；
(3) 调用`reset`方法，重置表单。



> 表单基类提供了输入字段（input field）的管理、验证、提交和表单加载服务。`Ext.form.Panel`类（`xtype: form`）以容器形式存在，并自动连接到一个`Ext.form.Basic`类的实例，这也是需要获取表单基类的引用以调用`reset`方法的原因。

看一下`onButtonClickCancel`方法中的参数：`button`、`e`和`options`，没有一个是表单的引用。

那么我们能做些什么呢？我们可以用`Button`类（继承自`AbstractComponent`类）的up方法。通过这个方法，可以用一个选择器获取表单。up方法回溯组件层级树，寻找匹配所传入选择器的祖先容器。

由于按钮在工具栏里，而工具栏又在我们要查找的表单里，使用`button.up('form')`，将

找到所需的结果。Ext JS将在组件层级树中寻找按钮的第一个祖先并找到工具栏，但工具栏并不是我们需要的，因此，它继续往上找，直至找到我们需要的表单。

在onButtonClickCancel方法中实现的代码如下：

```
button.up('form').getForm().reset();
```



一些开发者喜欢在window里实现toolbar，而不是在form里实现。这完全没有问题，只是个人喜好而已。在这种情况下，如果拥有提交按钮的工具栏包含在Window类里面，可以这么做：

```
button.up('window').down('form').getForm().reset()
```

这样也能达到同样目的！

2. 实现Submit按钮点击事件处理方法

现在，需要实现onButtonClickSubmit方法。在这个方法中，我们要完成把用户名和密码的值发送到服务器端以对用户进行验证的逻辑代码。

在这个方法里实现处理逻辑有两种选择：第一种是使用submit方法（表单基类提供），第二种是通过Ajax方式提交数据。无论哪种方式，都会达到目标。但是，在我们作出选择之前有一个细节必须给予关注：如果使用表单基类的submit方法，在把密码发送给服务器之前将无法将密码加密，查看一下发送给服务器端的参数，会发现密码是纯文本格式，这不是个好事儿。通过Ajax请求也将达到同样目标，而且我们可以在密码发送给服务器端之前将其加密。很显然，第二种选择看起来更合适，我们将采取这种方式。

方法执行的步骤如下。

- 获取Login表单引用。
- 获取Login窗体引用（一旦用户认证通过，我们就可以关闭它）。
- 通过表单获取用户名和密码。
- 加密密码。
- 发送登录信息至服务器端。
- 处理服务器端响应：
  - 如果用户通过认证，则呈现应用界面；
  - 如果用户未通过认证，则显示出错信息。

首先，我们来获取需要的引用：

```
var formPanel = button.up('form'),
```

```
    login = button.up('login'),
    user = formPanel.down('textfield[name=user]').getValue(),
    pass = formPanel.down('textfield[name=password]').getValue();
```

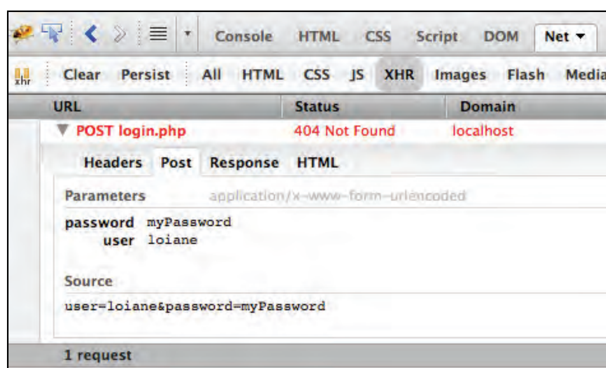
获取表单引用时，可以使用button.up('form')代码，就像曾经在onButtonClickCancel方法里使用的一样。采用同样的方式可获取登录窗体的引用，只需将选择器改为login或window，然后通过down方法获取user（用户名）和password（密码）字段的值，但现在选择用表单引用。用文本字段的xtype属性值作为选择器。为了确保获取正确的文本字段，可以创建一个itemId属性，但在这里不是必须的。我们可以采用name属性，因为user和password字段有不同的名称且它们在登录窗体里是唯一的。在选择器里使用属性需要把它包在方括号里。

下一步，提交值至服务器端：

```
if (formPanel.getForm().isValid()) {
    Ext.Ajax.request({
        url: 'php/login.php',
        params: {
            user: user,
            password: pass
        }
    });
}
```

如果试着运行这段代码，应用程序会发送请求至服务器端，但现在会得到一个响应错误，因为还未实现login.php代码。姑且把这个问题放一放，我们先来关注下别的细节。

打开Firebug或Chrome Developer Tools，再打开Net标签页并过滤出XHR请求信息，输入用户名和密码（确保输入了合法值，这样我们才可以正确提交），输出如下：



现在还未对密码进行加密，显示出来的还是原始值，这样并不好。我们需要对密码进行加密。

在app目录下，新建一个目录，命名为util，在这里将创建我们所需的工具类。创建一个新文件并命名为MD5.js，同时也有了一个名为Packt.util.MD5的新类。这个类包含了一个静态方法

encode，可以对所给的值进行MD5算法编码加密。要进一步了解MD5算法，请参考这里：http://en.wikipedia.org/wiki/MD5。因为Packt.util.MD5类的代码很多，就不在这里列出了，可以在http://www.packtpub.com/mastering-ext-javascript/book下载本书源代码，或到https://github.com/loiane/masteringextjs获取最新版本的源代码。



如果你想让密码传输变得更安全，也可以采用SSL加密技术，向服务器端请求一个随机盐[1]字符串，为密码加盐，并对其进行hash处理。读者可通过以下链接进一步掌握相关信息：http://en.wikipedia.org/wiki/Transport_Layer_Security和http://en.wikipedia.org/wiki/Salt_(cryptography)。

静态方法不通过类实例就能被调用。在Ext JS中，我们可以在静态配置里声明静态属性和静态方法。因为Packt.util.MD5类中的encode方法是静态方法，就可以像Packt.util.MD5.encode(value);这样来调用它。

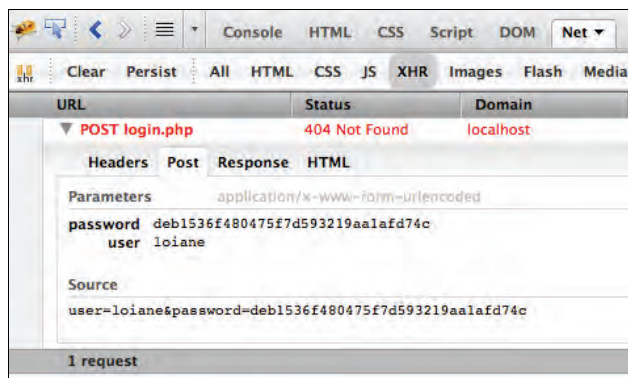
在Ext.Ajax.request之前，添加以下代码：

```
pass = Packt.util.MD5.encode(pass);
```

千万别忘了在控制器的requires（请求）声明当中添加Packt.util.MD5类（requires声明紧跟在extend声明后面）：

```
requires: [
    'Packt.util.MD5'
],
```

现在再一次运行代码，打开Net标签页观察XHR请求，输出如下图所示：

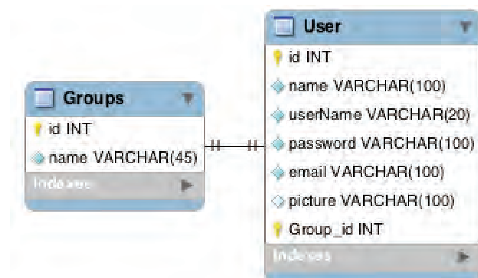


现在，密码经过加密变得足够安全了。

① 此处的“盐”其实就是一个随机生成的字符串。——译者注

2.4　创建用户和用户组表

在对login.php页面进行编码之前，我们还需要往sakila数据库中添加两张表。这两张表代表用户（User）和用户所属的用户组（Groups）。在这个项目里，用户只能归属于 一个用户组，关系如下图所示：



首先创建Groups表：

```
CREATE TABLE IF NOT EXISTS `sakila`.`Groups` (
  `id` INT NOT NULL AUTO_INCREMENT ,
  `name` VARCHAR(45) NOT NULL ,
  PRIMARY KEY (`id`) )
ENGINE = InnoDB;
```

接下来，我们创建带索引的User表，同时创建到Groups表的外键：

```
CREATE TABLE IF NOT EXISTS `sakila`.`User` (
  `id` INT NOT NULL AUTO_INCREMENT ,
  `name` VARCHAR(100) NOT NULL ,
  `userName` VARCHAR(20) NOT NULL ,
  `password` VARCHAR(35) NOT NULL ,
  `email` VARCHAR(100) NOT NULL ,
  `picture` VARCHAR(100) NULL ,
  `Group_id` INT NOT NULL ,
  PRIMARY KEY (`id`, `Group_id`) ,
  UNIQUE INDEX `userName_UNIQUE` (`userName` ASC) ,
  INDEX `fk_User_Group1_idx` (`Group_id` ASC) ,
  CONSTRAINT `fk_User_Group1`
    FOREIGN KEY (`Group_id` )
    REFERENCES `sakila`.`Groups` (`id` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

下一步，往这两张表里插入些数据：

```
INSERT INTO `sakila`.`Groups` (`name`) VALUES ('admin');
INSERT INTO `sakila`.`User` (`name`, `userName`, `password`, `email`, `Group_id`)
VALUES ('Loiane Groner', 'loiane', 'e10adc3949ba59abbe56e057f20f883e',
```

```
'me@loiane.com', '1');
```

因为密码被加密并保存在数据库当中，值e10adc3949ba59abbe56e057f20f883e解密后就是123456。

现在，我们准备开始编写login.php页面了。

2.5 服务器端的登录界面处理

由于我们有部分Ext JS代码需要发送登录信息到服务器端，因此，也需要实现对应的服务器端代码。如第1章里提到的，本书将用PHP实现服务器端代码。如果你不了解PHP，也不必担心，我们的PHP代码并不复杂，而且只使用PHP技术（不涉及各种PHP框架）。我们将重点关注服务器端程序处理逻辑，你也可以用自己喜欢的服务器端语言来实现同样的程序逻辑（Java、.NET、Python、Ruby等）。

2.5.1 连接数据库

第一步是创建负责连接数据库的PHP文件，我们后面要开发的PHP页面差不多都要重用这个文件。

在项目的根目录下创建名为php的文件夹，在php里面再创建名为db的文件夹，然后在db文件夹里创建名为db.php的文件：

```
<?php
$server = "127.0.0.1";
$user = "root";
$pass = "root";
$dbName = "sakila";

$mysqli = new mysqli($server, $user, $pass, $dbName);

/* 检查连接 */
if ($mysqli->connect_errno) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
?>
```

连接代码非常直接明了：只需告知服务器（本例中是localhost）、数据库用户名和密码，以及我们准备连接的数据库的名称。最后，检查连接操作成功与否。




若希望了解更多有关mysqli的信息，请访问http://php.net/manual/en/book.mysqli.php。



2.5.2　login.php

最后，我们在php文件夹里创建login.php文件，并实现它：

```
require("db/db.php"); // #1

session_start(); // #2

$userName = $_POST['user']; // #3
$pass = $_POST['password']; // #4

$userName = stripslashes($userName); // #5
$pass = stripslashes($pass);         // #6

$userName = $mysqli->real_escape_string($userName); // #7
$pass = $mysqli->real_escape_string($pass);         // #8
$sql = "SELECT * FROM USER WHERE userName='$userName' and
password='$pass'"; // #9
```

首先，我们要请求db.php文件以连接数据库（#1），然后开启一个会话，后面需把用户名保存在会话中（#2）。

下一步是取得`Ext.Ajax.request`发送的用户名和密码（#3和#4）。

`stripslashes`函数移除所获取字符串里的反斜杠（#5和#6）。比如，用户名是`Loiane\'s`，`stripslashes`函数的返回结果就是`Loiane's`。

然后用`real_escape_string`函数来准备在SQL语句中使用的`$username`和`$pass`变量（#7和#8）。`real_escape_string`函数用于转义SQL语句中所用字符串里的特殊字符[①]。

接下来，准备将被执行的SQL查询语句（#9）。这是一个简单的`SELECT`语句，返回一个与提供的用户名和密码条件相匹配的结果。

继续实现下一部分的代码：

```
$result = array(); // #10

if ($resultdb = $mysqli->query($sql)) { // #11

    $count = $resultdb->num_rows; // #12

    if($count==1){

        $_SESSION['authenticated'] = "yes"; // #13
```

① 转义特殊字符主要是出于安全方面的考量，防范“恶臭”的SQL注入攻击。——译者注

```
            $_SESSION['username'] = $userName; // #14

        $result['success'] = true; // #15
        $result['msg'] = 'User authenticated!'; // #16

    } else {

        $result['success'] = false; // #17
        $result['msg'] = 'Incorrect user or password.'; // #18
    }

    $resultdb->close(); // #19
}
```


login.php代码的第二部分里，首先创建了一个result变量（#10），存放我们打算返回给Ext JS的结果数据。

接下来，执行SQL查询语句，并存放结果到resultdb变量中（#11）。然后，存储结果集的行数（#12）。

现在到了最重要的代码部分，我们准备验证结果集返回了多少行。传入用户名和密码后，如果用户名和密码匹配上了数据库中的信息，结果集返回的行数应该正好是1。所以，如果行数是1的话，就将认证用户的用户名保存至会话中（#14），同时也保存用户通过认证的信息（#13）。

我们需要准备好返回给Ext JS的结果，主要返回两个信息：第一个是用户通过认证的信息（#15），本例中为true，同时再返回一个描述信息（#16）。

如果用户名和密码不匹配（结果集返回的行数不是1），我们也将返回一个信息给Ext JS告知用户名和密码不正确（#18），因此，success为false（#17）。然后，关闭结果集（#19）。

接下来是第三部分也是最后一部分login.php代码了：

```
$mysqli->close(); // #20

echo json_encode($result); // #21
```

关闭数据库连接（#20），把返回给Ext JS的结果编码成JSON（JavaScript Object Notation，JavaScript 对象表示法）数据格式（#21）。

现在，login.php页面的代码就完成了，最后别忘了把代码放在<?php和?>之间。

2.5.3　处理服务器端的返回结果——登录与否

我们已经了解了服务器端代码，现在回头来看Ext JS代码，并处理服务器端返回的响应信息。

但是，我们首先需要理解一个非常重要的概念，这个概念常常困扰着大多数的Ext JS开发者。

成功还是失败

`Ext.Ajax`类执行Ext JS发出的所有Ajax请求。如果我们看一下文档，就会发现这个类有三个事件，分别为：`beforerequest`、`requestcomplete`和`requestexception`。

`beforerequest`事件在请求发起前触发，`requestcomplete`事件在Ext JS准备获取服务器端响应时触发，而`requestexception`事件在服务器端返回HTTP错误状态时触发。

现在，我们回到`Ext.Ajax.request`的调用上。我们可以传一些选项给请求：想连接的URL、参数以及其他选项，包括success（成功）函数和failure（失败）函数。现在，误解出现了。一些开发者认为，如果服务器端动作成功完成，我们就会从服务器端返回`success = true`；如果不成功，我们就返回`success = false`。因此，`success = true`时的逻辑在success函数里处理，而`success = false`时的逻辑在failure函数里处理。这是错的，这不是Ext JS的工作机制。

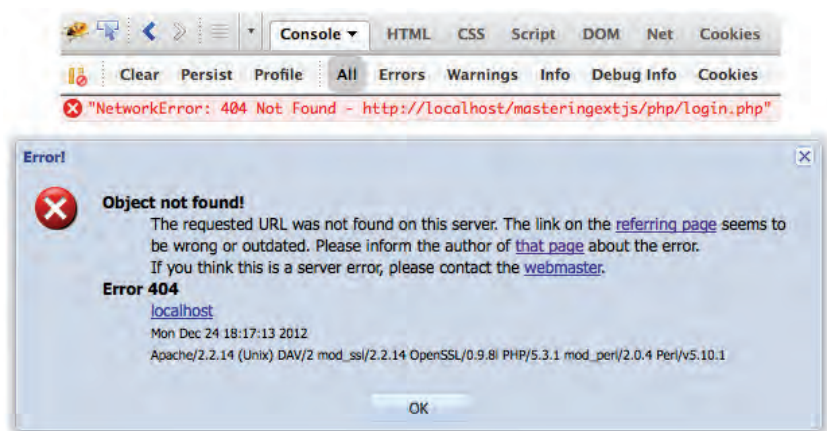
对于Ext JS而言，success表示服务器端返回了一个响应（`success = true`或是`success = false`都无所谓），failure表示服务器端返回了一个HTTP错误状态。这就意味着如果服务器端返回一个响应，我们就该在success函数里处理这个响应（同时，还需要处理`success`信息是`true`还是`false`的情况）；同样，在failure函数里需要通知用户信息出错了，用户据此联系系统维护人员。

我们先来实现`failure`函数的代码，在`Ext.Ajax.request`调用中，添加以下代码：

```
failure: function(conn, response, options, eOpts) {
    Ext.Msg.show({
        title:'Error!',
        msg: conn.responseText,
        icon: Ext.Msg.ERROR,
        buttons: Ext.Msg.OK
    });
}
```

结果将显示一个警告框给用户，警告框里有一个错误图标、一个OK按钮，以及HTTP错误状态信息。

我们来制造个错误以便请求异常事件被触发。出于测试目的，暂且更改login.php文件名（比如更改为login_.php），然后执行代码，输出如下：



这正是我们希望failure函数呈现的结果。项目所有Ext.Ajax.request调用里的所有failure函数都可重用这段代码。

现在我们来关注下success函数的代码实现：

```
success: function(conn, response, options, eOpts) {

    var result = Ext.JSON.decode(conn.responseText, true); // #1

    if (!result){ // #2
        result = {};
        result.success = false;
        result.msg = conn.responseText;
    }

    if (result.success) { // #3

        login.close(); // #4
        Ext.create('Packt.view.MyViewport'); // #5

    } else {
        Ext.Msg.show({
            title:'Fail!',
            msg: result.msg, // #6
            icon: Ext.Msg.ERROR,
            buttons: Ext.Msg.OK
        });
    }
},
```

首先我们要做的就是解码从服务器端收到的JSON信息（#1）。如果记录conn参数发送到success函数的信息（console.log(conn)），那么控制台输出将显示如下：

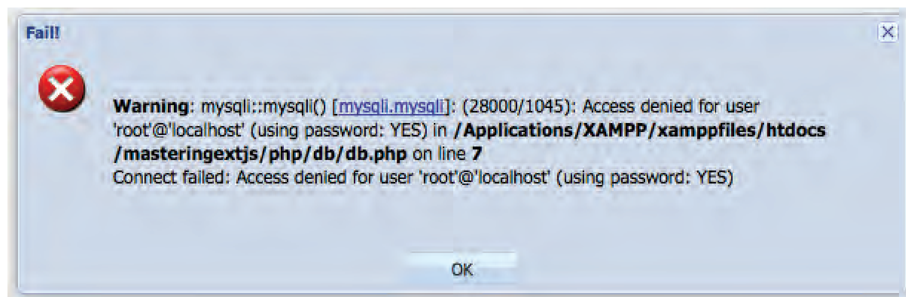
▶ request	Object { id=1, headers={...}, options={...}, more... }
requestId	1
▼ responseText	"{"success":false,"msg":"Incorrect user or password."}"
responseXML	null
status	200
statusText	"OK"
getAllResponseHeaders	function()
getResponseHeader	function()

解码conn.responseText，这正是我们希望取回的内容，这样就可以访问result.success和result.msg了。还需要注意一个细节：我们不知道服务器端会返回些什么，但我们总是期待返回的就是success和msg信息，然而，这并不能得到保证。如果其他MySQL错误信息被返回，它也将在conn.responseText里面并随之返回，那就不是我们期待的JSON数据格式了。如果出现这种情况，Ext.JSON.decode函数会失败并抛出异常。可以忽略这个异常（给Ext.JSON.decode函数的第二个参数传入true值，result变量的值就变为null了），但仍然需要处理它。这正是检查result变量是否为null时要做的工作（#2）。如果result为null，就实例化result变量并分配给它一些值（msg会接收服务器端发送的错误）。如果不进行处理，用户点击提交按钮将发现没有动静。

如果success为true（#3），就意味着用户在服务器端通过认证，那我们就可以做点什么事了，比如关闭登录窗体（#4），然后显示应用系统（#5）。因为现在还未创建Packt.view.MyViewport类，我们就先注释一下，等实现了该类再回来移除这个注释。在Packt.view.MyViewport类中，将显示菜单、标题、页脚，以及应用系统的中央面板，它是应用系统呈现界面的地方。

另外，当success为false时，意味着用户名或密码不匹配数据库里的信息，这时候就要显示一个服务器端发来的错误信息给用户（#6），表明输入了不正确的用户名或密码。

若服务器端发生了其他错误，就如我们在#2行描述的那样，错误会返回给Ext JS。举个例子，输入错误的密码给数据库连接代码，登录时就会显示如下错误：



如此一来，我们可以处理所有的服务器端响应，包括所有的异常，而这正是我们期待的。

2.6 优化登录界面

登录界面已经完成了。然而，我们还可以继续对它进行一些优化，让它变得更好，并提供更好的用户体验。

我们将对登录界面做以下优化：

- 认证进行时，提供一个加载遮罩；
- 用户按下回车键时提交表单；
- 显示信息提醒用户大写键已激活。

2.6.1 进行认证时为表单提供一个加载遮罩

有时候，用户点击提交按钮等待服务器端的响应，这个过程可能会有些延迟。有些用户可能很有耐心，但有些用户就没有，没耐心的用户会再次点击提交按钮，从而再一次发送请求到服务器端。我们可以在用户等待服务器端响应的同时，给登录窗体加载一个遮罩来规避这种行为。

首先，需要在Ext.Ajax.request调用代码前面添加以下代码：

```
Ext.get(login.getEl()).mask("Authenticating... Please wait...", 'loading');
```

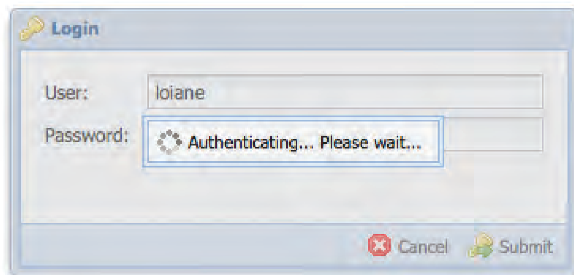
这将为登录窗体添加一个遮罩。

然后，在success和failure函数的第一行添加一行代码：

```
Ext.get(login.getEl()).unmask();
```

这将移除登录窗体上的遮罩。

如果现在执行代码，输出如下图所示：



请注意，这时所有的按钮都不能点击了，直到服务器端响应返回，用户才能再点击按钮。

2.6.2　回车提交表单

对于某些类型的表单，特别是登录表单，用户输入完信息后会很自然地敲个回车键。但Ext JS里并未自动提供这个功能，因此我们需要实现它。

textfield组件里有一个事件可以处理像回车键这样的特定按键。这个事件叫specialkey，就是我们在登录控制器里要监听的事件。首先，在this.control声明中加入以下事件代码：

```
"login form textfield": {
    specialkey: this. onTextfieldSpecialKey
}
```

我们使用了login form textfield选择器，从而可以获取登录表单中的所有文本字段，即user和password字段。之后用户在文本字段里按回车键就会提交表单。

接下来，同样是在控制器里实现onTextfieldSpecialKey方法：

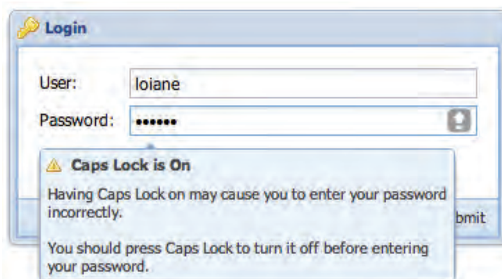
```
onTextfieldSpecialKey: function(field, e, options) {
    if (e.getKey() == e.ENTER){
        var submitBtn = field.up('form').down('button#submit');
        submitBtn.fireEvent('click', submitBtn, e, options);
    }
}
```

首先，我们需要验证用户按下的是否是回车键。然后，获取Submit按钮的引用：先获取表单的引用，再获取在组件层级中位于表单之下的Submit按钮的引用。最后，手动处理Submit按钮的点击事件触发。这种方式下，onButtonClickSubmit方法将自动被调用，因为仍在onButtonClickSubmit方法里验证表单的合法性，就可以确保客户端验证失败时请求不会被提交。

2.6.3　大写键提醒信息

最后的优化工作是给表单添加一个大写键提醒功能。大写键被激活的情况下输入密码时，由于系统区分大小写，即使输入正确的密码，系统仍会报错。所以，在此设个提醒是很有必要的。

激活大写键，输入信息后的最终效果如下所示：



从图中可以看到，我们通过工具提示（tooltip）显示提醒信息。因此，首先就是回到app.js的launch函数，在第一行添加以下代码：

```
Ext.tip.QuickTipManager.init();
```

没有这行代码，工具提示就无法工作。

另一个相关操作是在Ext.application（app.js）中把enableQuickTip属性设为true。

此外，需要监听keypress事件，我们只监听password字段触发的这个事件。默认状态下，textfield组件并不触发这个事件，因为这会对性能有所影响。现在要监听这个事件，就需要在password字段添加一个属性配置（在login.js文件中）。

```
name: 'password',
fieldLabel: "Password",
enableKeyEvents: true,
id: 'password'
```

同时，给password字段添加一个id。虽然我们曾经讨论过用id字段替代itemId是不好的，但在这里别无选择。因为，创建工具提示时，需要设置一个目标（这里，这个目标就是password字段），这个目标只能接受组件的id，而不接受itemId。

在添加代码到控制器之前，我们先创建工具提示。我们准备创建一个名为Packt.view.authentication.CapsLockTooltip的新视图，因此，需要在app/view/authentication目录下创建一个CapsLockTooltip.js文件。

```
Ext.define('Packt.view.authentication.CapsLockTooltip', {
    extend: 'Ext.tip.QuickTip',
    alias: 'widget.capslocktooltip',

    target: 'password',
    anchor: 'top',
    anchorOffset: 60,
    width: 300,
    dismissDelay: 0,
    autoHide: false,
    title: '<div class="capslock">Caps Lock is On</div>',
    html: '<div>Having Caps Lock on may cause you to enter your password</div>' +
        '<div>incorrectly.</div><br/>' +
        '<div>You should press Caps Lock to turn it off before entering</div>' +
        '<div>your password.</div>'
});
```

在Packt.view.authentication.CapsLockTooltip视图中声明一些属性配置来设置工具提示的行为。来看以下属性设置项。

- **target**　password字段自动的id值。

- **anchor**　表明提示信息固定在目标元素（password字段）的哪边[①]，带一个指回目标元素的小箭头。
- **achorOffset**　数值（以像素为单位），用来设置anchor箭头的偏移量，这里，箭头显示在离工具提示框起始位置60像素的地方。
- **width**　数值型（以像素为单位），表示工具提示框的宽度。
- **dismissDelay**　工具提示延迟隐藏的数值（以毫秒为单位），因为我们不希望工具提示自动隐藏，所以就将其设为0，禁止隐藏。
- **autoHide**　若设为true，当鼠标离开目标元素时工具提示自动隐藏，如果不希望这样的效果，就设为false。
- **title**　作为工具提示的标题文字。
- **html**　HTML文本片段，作为工具提示的显示信息。

在app.css中添加CSS代码（capslock样式类）：

```
.capslock{
    background:url('../icons/bullet_error.png') no-repeat center left;
    padding:2px;
    padding-left:20px;
    font-weight:700;
}
```

最后，在登录控制器里做些调整。首先，在视图（views）声明中，添加CapsLockTooltip类。因为我们在view目录里建了个子目录，如果只是简单地在控制器里添加一个CapsLockTooltip视图，控制器是不会识别的。因此，需要按照authentication.CapsLockTooltip这样的方式添加，控制器才能识别这个视图类是在view/authentication目录下。

```
views: [
    'Login',
    'authentication.CapsLockTooltip'
],
```

在app下的model、store、view以及controller目录里，会有很多我们需要的子目录。这可以帮助我们更好地组织代码，特别是创建大应用系统时。在视图、控制器、存储器以及模型声明中，也需要像authentication.CapsLockTooltip那样加上子目录名称。

下一步是在this.control声明中监听keypress事件：

```
"login form textfield[name=password]": {
    keypress: this.onTextfieldKeyPress
}
```

① 从截图可知，此处指password字段相对于工具提示的位置，比如本例中设为top，即password字段在工具提示上方。——译者注

我们要使用的选择器是login form textfield[name=password]。因为表单中有两个文本字段，那么就需要用某种方式找到password字段，我们可以通过name属性达到这个目的。

之后我们需要在控制器中实现onTextfieldKeyPress方法：


```
onTextfieldKeyPress: function(field, e, options) {
    var charCode = e.getCharCode(); // #1

    if((e.shiftKey && charCode >= 97 && charCode <= 122) || // #2
        (!e.shiftKey && charCode >= 65 && charCode <= 90)){

        if(this.getCapslockTooltip() === undefined){         // #3
            Ext.widget('capslocktooltip');                   // #4
        }

        this.getCapslockTooltip().show(); // #5
    } else {

        if(this.getCapslockTooltip() !== undefined){         // #6
            this.getCapslockTooltip().hide();                // #7
        }
    }
}
```

首先，获取用户按键的识别代码（#1）。然后验证是否按下Shift键的同时按下了小写字母键（a~z），或者Shift键没按下但按下了大写字母键（A~Z）（#2）。如果验证结果为ture，意味着大写键被激活。如果想了解每个键的识别代码，可以参考http://www.asciitable.com/。

如果大写键被激活，需要验证是否已存在 CapsLockTooltip类的引用（#3），如果不存在，就用它的xtype创建一个引用（#4）。

如果大写键未激活，也需要验证是否已存在CapsLockTooltip类的引用（#6），如果是，就隐藏工具提示。

最后一个细节：我们在控制器中并没有this.getCapslockTooltip()的引用。因此，需要创建它：

```
refs: [
    {
        ref: 'capslockTooltip',
        selector: 'capslocktooltip'
    }
]
```

ref（引用）是定位组件的另一选择，使用ComponentQuery（组件查询）语法。ref非常有用，特别是在一个控制器里面要多次获取一个组件的引用时。控制器会自动为一个ref生成一个get方法。在这里，控制器为我们生成了getCapslockTooltip方法。

现在完成大写键提醒功能了，我们可以保存项目并运行测试。

2.7　小结

在本章，我们一步步详细介绍了怎样创建登录界面。内容包括怎样根据Ext JS MVC架构创建登录视图和控制器，并组织代码结构。我们采用了表单的客户端验证，确保发送可接受的数据到服务器端，同时在发送密码到服务器端之前对其进行加密。本章还介绍了如何用PHP实现基本登录功能，以及怎样处理返回给Ext JS的服务器端数据。

我们还对登录（Login）界面进行了优化，如在用户按下回车键时提交表单，在`password`字段显示大写键锁定提醒，以及在发送数据到服务器端并等待服务器端返回数据的过程中，给表单添加加载遮罩。

下一章，我们将继续处理Login界面，了解怎样添加多语言功能，以及如何实现注销和会话监控功能。

第3章

注销与多语言支持

本章将实现系统的多语言支持功能。这个功能可将系统显示的标签内容翻译成用户选择的语言文字（其中会用到一些HTML5新特性）。

我们还将实现注销功能，以便用户可以结束会话。出于安全方面的考虑，我们还将实现交互过程中的会话超时警告功能（用户长时间未使用鼠标或键盘的情况）。

同时，用户认证通过后需要为其显示应用界面。本章学习如何使用视见区实现基本应用界面。

本章主要内容如下：

- 基本应用界面；
- 注销功能；
- 行为监控及会话超时警告；
- 为多语言支持组织应用结构；
- 创建语言转换组件；
- 实时处理语言转换。

3.1 基本应用界面

当我们在登录控制器的Submit按钮监听器里实现success函数时，提到过Packt.view.MyViewport类，现在就来实现它（在view目录下创建MyViewport.js新文件）。开始之前，先看一下输出界面，如下页所示。

生成上面所述图的代码如下：

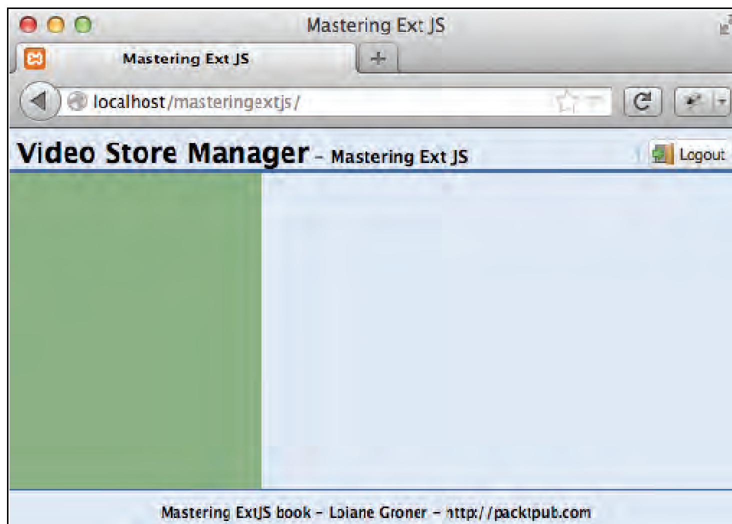
```
Ext.define('Packt.view.MyViewport', {
    extend: 'Ext.container.Viewport', // #1
    alias: 'widget.mainviewport',     // #2

    requires: [
        'Packt.view.Header' // #3
```

```
    ],

    layout: {
        type: 'border' // #4
    },

    items: [
        {
            xtype: 'container', // #5
            width: 185,
            collapsible: true,
            region: 'west',
            style: 'background-color: #8FB488;'
        },
        {
            xtype: 'appheader', // #6
            region: 'north'
        },
        {
            xtype: 'container', // #7
            region: 'center'
        },
        {
            xtype: 'container', // #8
            region: 'south',
            height: 30,
            style: 'border-top: 1px solid #4c72a4;',
            html: '<div id="titleHeader"><center><span
style="fontsize:10px;">Mastering ExtJS book - Loiane Groner -
http://packtpub.com</span></center></div>'
        }
    ]
});
```



Packt.view.MyViewport是一个视见区（#1），视见区是一个特殊容器，表示可视应用区域（浏览器视见区），将自身渲染到document body里，自动适配浏览器显示区域的大小并管理窗口缩放。一个页面只能有一个视见区。同时，我们还给这个类创建了一个别名（#2）。

MyViewport使用边界布局，边界布局划分成五个区域：上、下、左、右以及中央区域。中央区域是边界布局容器中唯一强制要求保留的区域。在本例中，我们不使用右边区域（#4）。

在items属性项数组中，声明了我们将用到的四个方位的组件。第一个组件是位于左边的容器（#5），为其设置宽度，并设置为可收缩形式，以便用户可以更好地查看中央区域。后续，我们将创建动态菜单并取代该容器。现在，先用浅绿色背景标识该区域。

第二个组件是appheader（#6），其xtype是Packt.view.Header类。在这个Header类里，将实现顶部区域界面功能，包括应用程序的标题以及注销按钮。因为使用了appheader这个xtype，所以要在类的requires声明里添加Packt.view.Header类（#3）。

第三个组件是中央区域容器（#7），这是后续显示应用界面的地方，我们用一个标签面板替代它。最后一个组件项是另一个容器，表示页脚（#8）。

现在，我们来实现Packt.view.Header类，在app/view目录里创建名为Header.js的新文件：

```
Ext.define('Packt.view.Header', {
    extend: 'Ext.toolbar.Toolbar', // #1
    alias: 'widget.appheader',     // #2

    height: 30, // #3
    ui: 'footer', // #4
    style: 'border-bottom: 4px solid #4c72a4;', // #5

    items: [
        {
            xtype: 'label', // #6
            html: '<div id="titleHeader">Video Store Manager<span
style="font-size:12px;"> - Mastering Ext JS</span></div>'
        },
        {
            xtype: 'tbfill' // #7
        },
        {
            xtype: 'tbseparator' // #8
        },
        {
            xtype: 'button', // #9
            text: 'Logout',
            itemId: 'logout',
            iconCls: 'logout'
        }
    ]
});
```

Header类扩展自Toolbar类（#1），给它分配别名（#2），我们在MyViewport类里用到了这个别名。然后，设置工具栏高度（#3），并设置一个样式效果，footer ui（#4）使得按钮在工具栏上看起来有突起效果，否则工具栏跟它的按钮看起来都是一样的颜色。同时，还给工具栏设置下边线样式（#5）。

接下来设置items属性项：第一个是标签组件，表示应用系统的标题（#6）。我们希望Logout（注销）按钮在工具栏的右侧，因此需要添加一个fill组件（#7）。然后，设置一个工具栏分割条，让工具栏看起来更美观些（#8）。最后是Logout按钮（#9）。

对#7行和#8行的代码，也可以应用快捷方式，所以以下代码：

```
{
    xtype: 'tbfill' // #7
},
{
    xtype: 'tbseparator' // #8
},
```

也可以这么写：

```
'->', // #7
'-', // #8
```

结果是一样的。这只是个人喜好，比如，对有经验的开发者而言，用快捷方式声明tbfill和tbseparator组件显然更高效，但还在进行代码维护的新手就需要积累更多关于快捷方式的经验。

最后，在app.css文件里添加一个新的样式：

```
#titleHeader {
    color:#000;
    font-size:20px;
    font-weight:bold;
    font-family:'Lucida Grande', Arial, Sans;
}
```

现在，我们有了基本的应用界面，后续章节里将不断对其进行完善。

3.2　注销功能

既然用户可以登录系统，那么他们也应该可以注销。注销是在登录控制器里最后实现的一个功能。

首先，在控制器的views属性项里添加Header类：

```
views: [
    'Login',
    'Header',
```

```
    'authentication.CapsLockTooltip'
]
```

接下来，在this.control声明里添加注销按钮的点击事件监听器：

```
"appheader button#logout": {
    click: this.onButtonClickLogout
}
```

作为选择器，我们使用appheader，因为它是Logout按钮所在的工具栏的xtype属性值，使用button是因为Logout按钮是按钮类型（xtype:button），#loginout使用了其对应的itemId值。这样可以确保ComponentQuery获取的按钮正是我们需要的。

现在，实现控制器的onButtonClickLogout方法：

```
onButtonClickLogout: function(button, e, options) {

    Ext.Ajax.request({
        url: 'php/logout.php', // #1
        success: function(conn, response, options, eOpts){

            var result = Ext.JSON.decode(conn.responseText, true);

            if (!result){
                result = {};
                result.success = false;
                result.msg = conn.responseText;
            }

            if (result.success) { // #3

                button.up('mainviewport').destroy(); // #4
                window.location.reload();            // #5
            }
            else {
                Ext.Msg.show({        // #6
                    title:'Error!',
                    msg: result.msg,
                    icon: Ext.Msg.ERROR,
                    buttons: Ext.Msg.OK
                });
            }
        },

        failure: function(conn, response, options, eOpts) {

                Ext.Msg.show({        // #7
                    title:'Error!',
                    msg: conn.responseText,
                    icon: Ext.Msg.ERROR,
```

```
                    buttons: Ext.Msg.OK
                });
            }
        });
    }
```

我们将实现一个到php/logout.php（很快就要创建它了）的Ajax调用（#1）。通常，需要处理success和failure这两个回调函数。在success函数里，首先解码服务器端响应。如果出错，就创建result变量的新实例来处理服务器端响应。如果成功（#3），获取mainviewport引用（Packt.view.MyViewport类），并销毁它（释放浏览器内存），同时，这样也将销毁所有应用组件（#4）。接下来，重新加载应用并回到登录界面（#5）。

如果success为false，就用出错信息显示一个错误警告框（#6）。

failure函数中，显示错误信息（#7）。

3.2.1　重构登录和注销代码

仔细观察发现，我们可以重用Submit按钮监听器的大量代码。这里“重用”一词并不贴切，实际上是代码复制，这并不好。我们如何维护代码并只进行简单调整呢？改动所有需要改动的地方显然是件枯燥的事情。这就需要重构代码，并按可重用的方式创建代码。

因此，我们在util目录下创建一个新的Util类：Packt.util.Util。

```
Ext.define('Packt.util.Util', {

    statics : { // #1

        decodeJSON : function (text) { // #2

            var result = Ext.JSON.decode(text, true);

            if (!result){
                result = {};
                result.success = false;
                result.msg = text;
            }

            return result;
        },

        showErrorMsg: function (text) { // #3

            Ext.Msg.show({
                title:'Error!',
                msg: text,
```

```
                icon: Ext.Msg.ERROR,
                buttons: Ext.Msg.OK
            });
        }
    }
});
```

所有的方法或函数都是静态的（#1），从而避免创建类实例。

首先声明decodeJSON方法（#2），调用decode方法解码服务器端响应，并处理因出错导致无法解码服务器端响应的情况。实现的第二个方法是showErrorMsg（#3），简单显示一个带有OK按钮和出错图标的错误警告框。

3

回到登录控制器，在requires声明中添加Packt.util.Util类。接下来，重构onButtonClickLogout方法：

```
onButtonClickLogout: function(button, e, options) {

    Ext.Ajax.request({
        url: 'php/logout.php',
        success: function(conn, response, options, eOpts){

            var result =
            Packt.util.Util.decodeJSON(conn.responseText);

            if (result.success) {

                button.up('mainviewport').destroy();
                window.location.reload();
            }
            else {
                Packt.util.Util.showErrorMsg(conn.responseText);
            }
        },
        failure: function(conn, response, options, eOpts) {
            Packt.util.Util.showErrorMsg(conn.responseText);
        }
    });
}
```

我们做了大量清理，只关注跟注销相关的代码。代码还能进一步优化，但就目前而言这已经足够了。也可以对onButtonClickSubmit方法进行类似重构。



> 重构看起来好像没有必要，但它是**有效载荷尺寸最小化**（minimizing the payload size，JavaScript开发过程中的一个最佳实践，也是Web开发关注话题）的一部分。可通过以下链接了解更多关于有效载荷尺寸最小化的内容：https://developers.google.com/speed/docs/best-practices/payload。

3.2.2　服务器端注销功能

我们在php目录下创建一个名为logout.php的PHP新页面，用来实现服务器端的注销功能，其代码非常简单：

```
<?php

session_start(); // #1

$_SESSION = array(); // #2

session_destroy(); // #3

$result = array(); // #4

$result['success'] = true;
$result['msg'] = 'logout';

echo json_encode($result); // #5

?>
```

首先重启会话（#1），之后释放所有会话变量（#2）、销毁会话（#3）。最后，返回信息给Ext JS告知会话被销毁（#4、#5）。

这样，我们就实现了服务器端注销功能。

3.2.3　客户端行为监控

让我们进一步丰富应用的功能。有一点很重要，那就是告知用户Web应用有超时设置，出于安全考虑不能在用户离开的情况下保持长时间登录状态。服务器端语言能够实现超时控制，一旦用户登录，出于安全考虑，服务器端不可以无条件地一直“记着”他，这也是添加这个功能的原因。

我们考虑用一个插件来实现此功能。这个插件叫`Packt.util.SessionMonitor`，是基于Sencha市场的`Activity Monitor`插件（https://market.sencha.com/extensions/extjs-activity-monitor）。一段时间后（默认是15分钟未操作），插件会显示信息询问用户是否要继续保持激活。如果是，插件发送一个Ajax请求到服务器端，服务器端将维持会话；如果在信息显示的60秒内用户未作出反应，应用就自动注销。

读者可以从https://github.com/loiane/masteringextjs/blob/master/app/util/SessionMonitor.js获取插件源代码。

如果想改变未进行操作的时间间隔，只需改变`maxInactive`设置即可。

要开始监控会话，需在登录控制器onButtonClickSubmit方法的MyViewport类实例化代码之后添加以下几行代码：

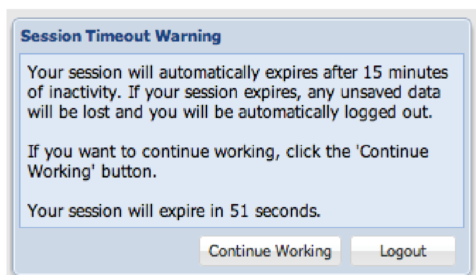
```
Ext.create('Packt.view.MyViewport');
Packt.util.SessionMonitor.start();
```

SessionMonitor.js的第42行，调用了一个名为php/sessionAlive.php的文件。我们需要在php目录下创建这个文件，并实现如下代码：

```
<?php
session_start();
?>
```


上述实现仅维持了服务器端会话并重设15分钟计时间隔。

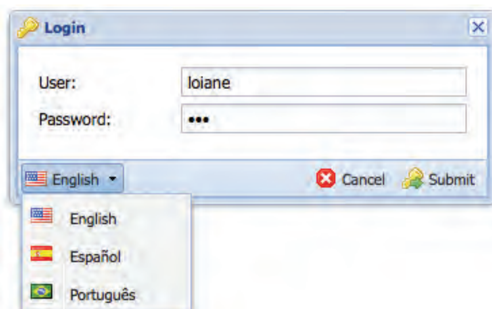
运行程序并保持15分钟的无操作状态，可以看到以下截图信息：



Ext JS并未自带此功能，但如我们所见，很容易实现该功能并将插件应用于其他Ext JS项目。

3.3　多语言支持

我们的产品很可能会远销海外，这时候系统具备语言转换能力就显得很重要了，毕竟不是所有用户都使用同一种语言。本节，我们就来实现一个具备标签语言转换功能的多语言组件。本节功能完成时，界面将如下所示：



我们的想法是在本地存储用户语言偏好，以便下一次用户加载应用程序时，自动显示首选语言。而当用户改变语言时，应用程序需要被重新加载，以使新的翻译可以加载到存储器中。

3.3.1　创建语言转换组件

仔细看本节开头的图，就会注意到语言转换组件是个按钮，点击箭头会弹出一个带有语言可选项的菜单。

带有箭头的按钮叫分割按钮（Split button）组件，它有个菜单，每种语言作为一个菜单（Menu）项。接下来，创建一个Packt.view.Translation类，包含我们描述的这些特性。在app/view目录下创建一个名为Translation.js的新文件：

```
Ext.define('Packt.view.Translation', {
    extend: 'Ext.button.Split', // #1
    alias: 'widget.translation', // #2
    menu: Ext.create('Ext.menu.Menu', { // #3
        items: [
            {
                xtype: 'menuitem', // #4
                iconCls: 'en',
                text: 'English'
            },
            {
                xtype: 'menuitem', // #5
                iconCls: 'es',
                text: 'Español'
            },
            {
                xtype: 'menuitem', // #6
                iconCls: 'pt_BR',
                text: 'Português'
            }
        ]
    })
});
```

创建的类扩展自Splitbutton（#1）。Splitbutton类提供了一个内置下拉箭头，并可触发与默认按钮点击事件不同的事件。典型的就是显示一个下拉菜单，提供给主按钮更多可选项。我们分配一个别名给它（#2）。

在menu属性配置项里创建一个Menu类的实例（#3），items里是每种转换语言选项，有：转换成英语的选项（#4），显示美国旗标；转换成西班牙语的选项（#5），显示西班牙旗标；转换成葡萄牙语的选项（#6），显示巴西旗标。

还可以添加我们需要的各种语言选项，对于每一种转换语言选项，只需要添加一个新的菜单项。

下一步，在app.css中为iconCls添加添加CSS样式：

```
.pt_BR {
    background-image:url('../flags/br.png') !important;
}

.en {
    background-image:url('../flags/us.png') !important;
}

.es {
    background-image:url('../flags/es.png') !important;
}
```


组件现在初具雏形（只能展现给用户而已）。我们会在项目的两个地方使用了语言转换组件：登录界面以及顶部区域工具栏的Logout按钮前面。

首先，在登录界面添加这个组件。再次打开Packt.view.Login类，作为工具栏第一个子组件项添加这个组件，使其看起来像本节开头图示那样：

```
items: [
  {
    xtype: 'translation'
  },
  {
    xtype: 'tbfill'
  },
  //
]
```

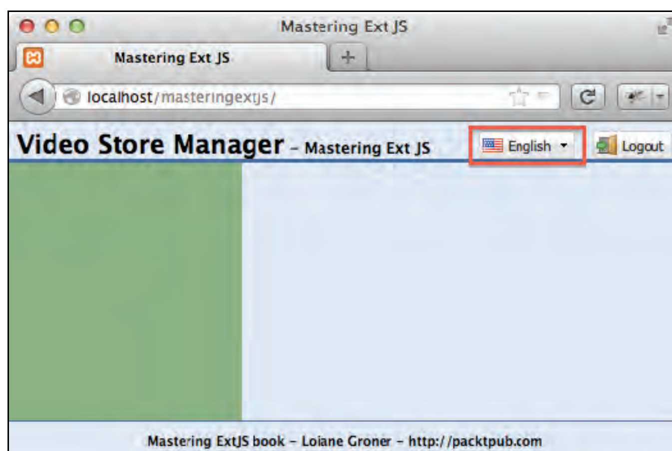
别忘了在登录界面类的requires声明里添加这个类：

```
requires: [
    'Packt.view.Translation'
],
```

接下来在Packt.view.Header类里进行同样的工作。在工具栏fill组件后面添加语言转换组件。

```
{
    xtype: 'tbfill'
},
{
    xtype: 'translation'
},
{
    xtype: 'tbseparator'
},
```

如果我们这时候运行程序，可以看到登录界面有语言转换组件了，登录后，在Logout按钮的前面也有这个组件：



3.3.2　创建转换文件

我们需要在项目中保存转换信息，分别保存每一种语言的转换信息到相应JavaScript文件里，并放在translations目录下。由于打算用`iconCls`的值加载转换文件，就需要创建3个名为en.js、es.js以及pt_BR.js的文件。每个文件中，都将创建一个名为`translations`的JavaScript对象，这个对象的每个属性就是一条转换信息，所有的转换文件都是这样，唯一不同的是转换信息的内容。

例如，以下是en.js的代码：

```
translations = {
    login: "Please Login",
    user: "User",
    password: "Password",

    cancel: "Cancel",
    submit: "Submit",
    logout: 'Logout',

    capsLockTitle: 'Caps Lock is On',
    capsLockMsg1: 'Having Caps Lock on may cause you to enter your password',
    capsLockMsg2: 'incorrectly.',
    capsLockMsg3: 'You should press Caps Lock to turn it off before entering',
    capsLockMsg4: 'your password.'
}
```

以下是pt_BR.js代码，包含了葡萄牙语的转换信息：

```
translations = {
    login: "Faça o Login",
    user: "Usuário",
    password: "Senha",

    cancel: "Cancelar",
```

```
        submit: "Enviar",
        logout: 'Logout',

        capsLockTitle: 'Caps Lock está ativada',
        capsLockMsg1: 'Se Caps lock estiver ativado, isso pode fazer com que você',
        capsLockMsg2: 'digite a senha incorretamente.',
        capsLockMsg3: 'Você deve pressionar a tecla Caps lock para desativá-la',
        capsLockMsg4: 'antes de digitar a senha.'
    }
```


文件格式都一样，仅是转换信息不同。随着应用的完善，将不断添加转换信息，这是一种组织文件的好方式，方便我们后续更改转换信息。

3.3.3 使用转换信息

要在组件中应用这些转换信息，我们还需继续努力，用`translation.property`方式替代将呈现在标签上的字串。

比如，`Packt.view.Login`类里有窗体的标题、用户名和密码的标签信息（`fieldLabel`），以及Cancel和Submit按钮的显示标签。要将转换文件中的转换信息应用到这些标签上面没这么容易。

用以下代码替代登录窗体的`title`设置：

```
title: translations.login,
```

用以下代码替代用户名字段的`fieldLabel`设置：

```
fieldLabel: translations.user,
```

用以下代码替代用密码字段的`fieldLabel`设置：

```
fieldLabel: translations.password,
```

用以下代码替代用Cancel按钮的`text`设置：

```
text: translations.cancel
```

用以下代码替代用Submit按钮的`text`设置：

```
text: translations.submit
```

以此类推，我们也可以为Logout按钮，乃至`CapsLockTooltip`类应用转换信息。

3.3.4 HTML5 本地存储

我们的转换组件实现思路是为用户存储一些语言偏好（指转换信息）。可以用cookie来实现，但这想法并不成熟，cookie需要依赖HTTP请求。我们想要长期保存这些信息，并寻求一种不受页

面刷新甚至关闭浏览器影响的实现方式。这种完美的方式就是使用本地存储（Local Storage）——HTML5的一个新特性。

Ext JS已通过`LocalStorage`代理实现了对本地存储的支持。我们想要更简单的方式，而在编码中使用HTML5特性本就是一种简便化方式，同时也能论证一下其他APIs跟Ext JS API的协作性。

并非所有浏览器都支持本地存储，提供支持的浏览器有：IE 8.0+、Firefox 3.5+、Safari 4.0+、Chrome 4.0+、Opera 10.5+、iPhone 2.0+以及Android 2.0+。后面会创建一个友好页面提示用户升级浏览器。我们也会在其他界面中融合应用Ext JS和更多的HTML5特性。现在，我们需要了解待实现的代码并非在所有浏览器里都能运行。

若想了解更多关于HTML5存储的信息，请访问http://diveintohtml5.info/storage.html。

在translations目录里创建名为locale.js的新文件，实现代码如下：

```
var lang = localStorage ? (localStorage.getItem('user-lang') || 'en') : 'en';
var file = 'translations/' + lang + '.js';

document.write('<script type="text/javascript" src="' + file + '"></script>');
```

首先，判断`localStorage`代理是否可用。如果可用，则判断在`localStorage`中是否有名为user-lang的项，如果该项不存在则默认使用英语。如果`localStorage`不可用，也默认使用英语。

然后，创建`file`变量，用来放置应用需加载的转换文件的路径。

最后，我们把选择的转换文件加到index.html页面中去，需在index.html文件里添加以下代码：

```
<script src="translations/locale.js"></script>
```

这行代码添加我们创建的locale.js。当浏览器加载 index.html页面时，脚本将被执行，应用加载后，多语言支持功能就可以使用了。

3.3.5　实时的语言切换

现在转换组件的实现到了尾声，当用户选择不同语言时，需要重新加载整个应用以使`translations/locale.js`再次执行，并加载相应的新语言。

我们需要创建一个新的控制器处理转换组件，这个新的控制器类名为`Packt.controller.TranslationManager`，因此，在app/controller目录下创建名为TranslationManager.js的新文件。

在这个控制器中，需监听两个事件：一个由转换组件自身触发，另一个由菜单项触发。

```
Ext.define('Packt.controller.TranslationManager', {
    extend: 'Ext.app.Controller',

    views: [
        'Translation' // #1
    ],

    refs: [
        {
            ref: 'translation', // #2
            selector: 'translation'
        }
    ],

    init: function(application) {
        this.control({
            "translation menuitem": { // #3
                click: this.onMenuitemClick
            },
            "translation": {          // #4
                beforerender: this.onSplitbuttonBeforeRender
            }
        });
    }
});
```

首先，在views声明转换组件（#1），作为控制器的视图，我们需要控制器负责转换组件的所有事件触发。然后，创建转换组件的引用，转换组件的xtype为translation，所以命名ref为translation（#2）。有了ref声明，控制器会自动产生getTranslation方法，通过它可以获取转换组件的引用。

下一步就开始监听我们感兴趣的事件了。首先，需要监听转换组件的beforerender事件（#4），转换组件是一个Split（分割）按钮。beforerender事件在分割按钮被渲染前触发。我们想要的动作是基于用户的语言选择，并设置用户所选语言的图标以及名称。第二个监听事件是菜单项的点击（#3）。当用户点击某个菜单选项时，我们需要设置事先在localStorage里的新语言，改变Split按钮的图标和文本，并重新加载应用。

先看看onSplitbuttonBeforeRender方法：

```
onSplitbuttonBeforeRender: function(abstractcomponent, options) {
    var lang = localStorage ? (localStorage.getItem('user-lang') || 'en') : 'en'; //
#5

    abstractcomponent.iconCls = lang; // #6
    if (lang == 'en') { // #7
        abstractcomponent.text = 'English'; // #8
```

```
        } else if (lang == 'es'){
            abstractcomponent.text = 'Español';
        } else {
            abstractcomponent.text = 'Português';
        }
    }
```

首先判断浏览器是否支持localStorage，如果支持，则可以操作语言本地存储特性。如果不支持，或者语言偏好尚未设置（用户初次使用应用或用户尚未改变语言的情况下），默认语言为英语（#5）。然后，设置分割按钮的iconCls为所选语言对应的旗标（#6）。

如果选择语言是英语，则设置Split按钮文本为“English”（#7）；西班牙语则设置为“Español”（#8）；葡萄牙语则设置为“Português”。

现在来看onMenuitemClick方法：

```
onMenuitemClick: function(item, e, options) {
    var menu = this.getTranslation(); // #9

    menu.setIconCls(item.iconCls); // #10
    menu.setText(item.text);       // #11

    localStorage.setItem("user-lang", item.iconCls); // #12

    window.location.reload(); // #13
}
```

首先，获取translation组件的引用（#9）。然后，设置Split按钮的iconCls和菜单项文本（#10和#11）。下一步，设置用户选择的对应localStorage里的新语言（#12），最后重新加载页面（#13）。

别忘了在app.js文件的controllers声明中添加一个新的控制器：

```
controllers: [
    'Login',
    'TranslationManager'
],
```

执行程序并更改偏好语言，我们可以发现界面语言也随之发生转变：

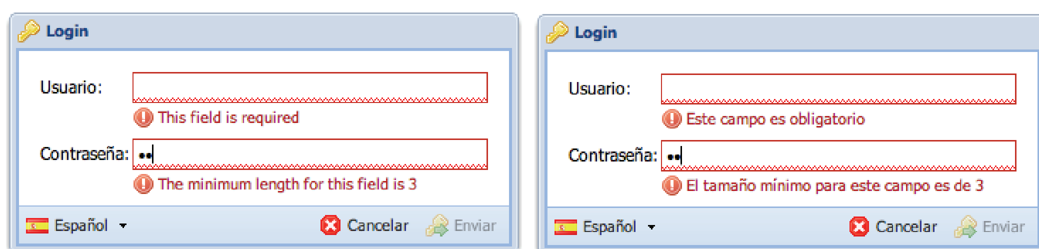


3.3.6　本地化：Ext JS语言转换

还漏了一件事，我们只转换了应用系统的标签信息。作为Ext JS API一部分的错误信息或其他信息并没有转换。但Ext JS提供了本地化支持，要做的仅仅只是把JavaScript的locale文件（本地化文件）添加到HTML页面。把以下代码添加到translations/locale.js文件中：

```
var extjsFile = 'extjs/locale/ext-lang-' + lang + '.js';
document.write('<script type="text/javascript" src="' + extjsFile + '"></script>');
```

试着再次执行程序，就会发现所有的Ext JS信息都被转换了。例如，如果转换语言为西班牙语，表单的验证错误信息也将转换成西班牙语：



现在，应用系统的转换功能全部完成了！当用户改变语言选择时，重新加载应用的原因之一在于为了重新加载正确的本地化文件。

3.4　小结

我们实现了本书应用案例的基本功能，并实现了注销按钮（Ext JS客户端实现以及服务器端实现）。同时，我们还实现了行为监控以及会话超时控制功能。

最后，我们掌握了如何融合HTML5特性与Ext JS创建语言转换组件，以实时转换应用程序的标签信息及偏好语言。

本章我们进一步丰富了应用程序的功能，新文件越来越多，应用也越来越复杂。后续各章还会创建更多的文件和组件。

下一章，我们将学习怎样用折叠面板和树形面板创建动态菜单。

第 4 章

动态菜单高级应用

4

在第2章中，我们已经实现了登录功能，现在需要实现主界面功能。首先要实现的是动态菜单功能：用户权限不同，在菜单中展现出来的菜单项也不同。一旦用户通过了认证，将进入应用系统的主界面，主界面由采用框线分隔布局的视见区构成，在视见区左边将展示一个菜单。

在界面渲染上，可以渲染所有的系统界面元素，然后根据不同的用户角色，隐藏或显示这些界面元素。但在本书中，我们并不采用这种方式，只渲染和展示用户能访问的那些界面元素，方法就是根据用户的访问权限动态渲染菜单。

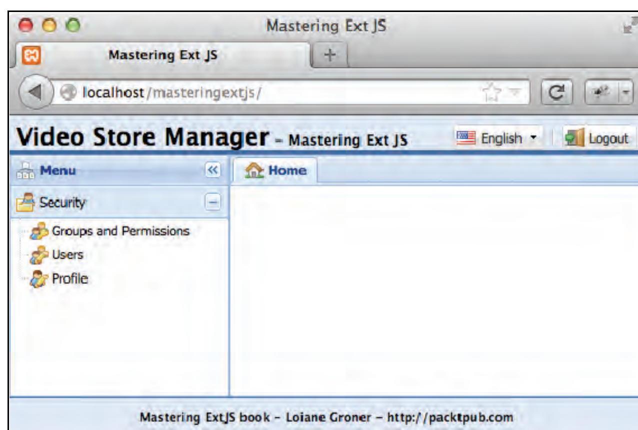
我们将学会通过折叠面板（Accordion panel）和树形面板（Tree panel）展示动态菜单——一个更复杂的动态菜单。本章介绍以下内容：

- 通过折叠面板和树形面板实现动态菜单；
- 通过hasMany绑定从服务器端加载数据；
- 在服务器端处理动态菜单；
- 动态打开菜单项。

4.1 创建动态菜单

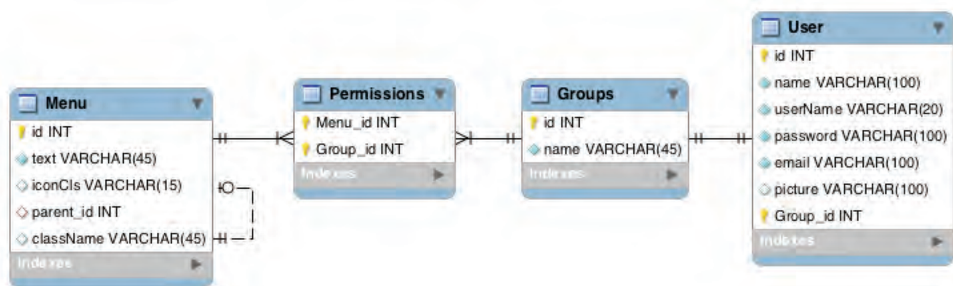
本章创建的第一个组件是动态菜单。只用树形面板也可以展示菜单，但我们希望干些有挑战性的工作，用折叠面板和树形面板实现一个增强版动态菜单，从而给用户更好的体验。系统由各个模块构成，每个模块又有若干子项，点击子项呈现对应的界面。折叠面板用来呈现这些模块，通过它，用户可逐一展开每个模块并看到其子项（菜单项）。每个模块的子项，用树形面板来实现，树节点就是每个子项。

最终，我们的动态菜单如下图所示：



4.1.1　数据库模型：用户组、菜单及权限

我们已经创建了User表和Groups表，为了存储菜单和菜单项的内容，以及每个用户组的对应权限，需要再创建两个数据库表，即Menu表和Permissions表，如下图所示：



Menu表里存储所有的菜单信息，每一个菜单项都将作为树形面板里的树节点，这就要求信息的存储形式应当与树形面板适配。因此，Menu表里：id字段表示树节点；text字段表示树节点显示的文字描述（在本例中，因为应用了多语言支持功能，将存储语言转换文件的属性）；iconCls字段表示树节点图标要用的CSS类；className字段表示将动态实例化的类的alias（或xtype）属性，实例化后的界面组件呈现在系统中部位置标签面板里；parent_id字段表示父节点[①]（有时没有该字段，模块是没有parent_id的，而模块的子项有parent_id，即模块）。

此外，由于Menu表与Groups表之间是N:N（多对多）的关系，我们设计了一个Permissions表来描述这种关系。下一章会进一步讨论如何分配用户到用户组中去。

我们用下列SQL脚本创建Menu和Permissions这两个新表：

① 原文是根节点，应该是父节点比较合适，只不过本例中的父节点即是根节点，因为只有二级层次。——译者注

```
USE `sakila` ;
CREATE  TABLE IF NOT EXISTS `sakila`.`Menu` (
  `id` INT NOT NULL AUTO_INCREMENT ,
  `text` VARCHAR(45) NOT NULL ,
  `iconCls` VARCHAR(15) NULL ,
  `parent_id` INT NULL ,
  `className` VARCHAR(45) NULL ,
  PRIMARY KEY (`id`) ,
  INDEX `fk_Menu_Menu` (`parent_id` ASC) ,
  CONSTRAINT `fk_Menu_Menu`
    FOREIGN KEY (`parent_id` )
    REFERENCES `sakila`.`Menu` (`id` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
CREATE  TABLE IF NOT EXISTS `sakila`.`Permissions` (
  `Menu_id` INT NOT NULL ,
  `Group_id` INT NOT NULL ,
  PRIMARY KEY (`Menu_id`, `Group_id`) ,
  INDEX `fk_Menu_has_Group_Group1` (`Group_id` ASC) ,
  INDEX `fk_Menu_has_Group_Menu1` (`Menu_id` ASC) ,
  CONSTRAINT `fk_Menu_has_Group_Menu1`
    FOREIGN KEY (`Menu_id` )
    REFERENCES `sakila`.`Menu` (`id` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Menu_has_Group_Group1`
    FOREIGN KEY (`Group_id` )
    REFERENCES `sakila`.`Groups` (`id` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

接下来，需要给Menu表和Permissions表填充一些数据。为了能够顺利完成后续的安全模块，还需要生成3个菜单项，现在，className先设为“panel”（表示是一个panel类型），以方便测试。

```
INSERT INTO `sakila`.`menu` (`id`, `text`, `iconCls`)
VALUES (1, 'menu1', 'menu_admin');

INSERT INTO `sakila`.`menu` (`id`, `text`, `iconCls`, `parent_id`, `className`)
VALUES
(2, 'menu11', 'menu_groups', 1, 'panel'),
(3, 'menu12', 'menu_users', 1, 'panel');

INSERT INTO `sakila`.`Permissions` (`Menu_id`, `Group_id`)
VALUES ('1', '1'), ('2', '1'), ('3', '1');
```

4.1.2　创建菜单模型：**hasMany**绑定

现在可以开始编码了。首先，建立一个从服务器端加载菜单数据的模型类，类名为：

Packt.model.menu.Root。在app/model目录下新建一个子目录menu，然后创建一个名为Root.js的文件，Packt.model.menu.Root类实现代码如下所示：

```
Ext.define('Packt.model.menu.Root', {
    extend: 'Ext.data.Model',

    uses: [
        'Packt.model.menu.Item'
    ],

    idProperty: 'id',

    fields: [
        {
            name: 'text'
        },
        {
            name: 'iconCls'
        },
        {
            name: 'id'
        }
    ],

    hasMany: {
        model: 'Packt.model.menu.Item',
        foreignKey: 'parent_id',
        name: 'items'
    }
});
```

需要为这个类提供的信息有：面板标题（text）、iconCls和id等属性。Root类有一个hasMany绑定类Packt.model.menu.Item，表示树形面板的每个树节点。设置hasMany绑定的name属性为items，Ext JS会创建一个名称为items的方法，这样就可以取回menu.Item类的集合。现在来实现Packt.model.menu.Item类：

```
Ext.define('Packt.model.menu.Item', {
    extend: 'Ext.data.Model',

    uses: [
        'Packt.model.menu.Root'
    ],

    idProperty: 'id',

    fields: [
        {
            name: 'text'
        },
        {
            name: 'iconCls'
        },
```

4

```
        {
            name: 'className'
        },
        {
            name: 'id'
        },
        {
            name: 'parent_id'
        }
    ],

    belongsTo: {
        model: 'Packt.model.menu.Root',
        foreignKey: 'parent_id'
    }
});
```

这个类表示树形面板的某个树节点，根据Menu表的字段声明fields属性，因此，对应的字段有：text、iconCls、className、id和parent_id。parent_id字段对应menu.Root的外键。menu.Root类有个hasMany绑定，而menu.Item类有个逆向作用的belongsTo绑定：menu.Item类通过它取回menu.Root类。

有了这些模型（menu.Root和menu.Item）和绑定（hasMany绑定和belongsTo绑定），就可以从服务器端获取嵌套数据，以下的JSON格式数据，表示从menu.Root类和menu.Item类获取的数据：

```
"items": [{
    "id": "1",
    "text": "menu1",
    "iconCls": "menu_admin",
    "parent_id": null,
    "className": null,
    "leaf": false,
    "items": [{
        "id": "2",
        "text": "menu11",
        "iconCls": "menu_groups",
        "parent_id": "1",
        "className": "panel",
        "leaf": true
    }, {
        "id": "3",
        "text": "menu12",
        "iconCls": "menu_users",
        "parent_id": "1",
        "className": "panel",
        "leaf": true
    }, {
        "id": "4",
        "text": "menu13",
        "iconCls": "menu_profile",
```

```
            "parent_id": "1",
            "className": "panel",
            "leaf": true
        }]
    }]
```

4.1.3 创建数据存储器：通过服务器端加载菜单

现在已经创建了模型，接下来创建数据存储器。创建一个名为`Packt.store.Menu`的新类，相应的，在app/store目录下创建一个名为Menu.js的文件：

```
Ext.define('Packt.store.Menu', {
    extend: 'Ext.data.Store',

    requires: [
        'Packt.model.menu.Root'
    ],

    model: 'Packt.model.menu.Root',

    proxy: {
        type: 'ajax',
        url: 'php/menu.php',

        reader: {
            type: 'json',
            root: 'items'
        }
    }
});
```

这个数据存储器使用`Packt.model.menu.Root`模型，并且使用Ajax代理方式，这意味着可以通过提供的`url`发送Ajax请求到服务器端，通过名为`items`的`Root`类获取如我们前面讨论模型时列出的JSON格式数据。现在已经知道了需从服务器端获取数据的格式，接下来，就要实现数据获取功能。

4.1.4 在服务器端处理动态菜单

参考我们在`Packt.store.Menu`数据存储器代码里的描述，在php目录下创建一个新文件menu.php。并按以下程序逻辑处理：

(1) 打开数据库连接；
(2) 通过会话获取登录用户信息；
(3) 从`Permission`表获取菜单的ID，这样才知道用户对应的权限；
(4) 获取用户具备权限的模块，`parent_id`为`Null`；
(5) 对每个模块，获取用户可访问的节点（菜单项）；
(6) 编码从服务器端返回的JSON格式数据；

(7) 关闭数据库连接。

开始编写代码：

```
require("db/db.php"); // #1

session_start(); // #2

$username = $_SESSION[username]; // #3

$queryString = "SELECT p.menu_id menuId FROM User u "; // #4
$queryString .= "INNER JOIN permissions p ON u.group_id= p.group_id ";
$queryString .= "INNER JOIN menu m ON p.menu_id= m.id ";
$queryString .= "WHERE u.username = '$username' ";

$folder = array(); // #5
```

首先，打开数据库连接（#1）。然后开启会话（#2），通过会话获取认证用户的username信息（#3）。准备SQL查询语句，获取用户有相应访问权限的Menu表的ID字段（#4）。最后，声明并初始化返回给Ext JS的变量（#5）。

继续编写代码：

```
if ($resultdb = $mysqli->query($queryString)) { // #6

    $in = '('; // #7
    while($user = $resultdb->fetch_assoc()) {
        $in .=  $user['menuId'] . ","; // #8
    }
    $in = substr($in, 0, -1) . ")"; // #9

    $resultdb->free();  // #10

    $sql = "SELECT * FROM MENU WHERE parent_id IS NULL ";
    $sql .= "AND id in $in"; // #11
    if ($resultdb = $mysqli->query($sql)) { // #12

        while($r = $resultdb->fetch_assoc()) { // #13

            $sqlquery = "SELECT * FROM MENU WHERE parent_id= '";
            $sqlquery .= $r['id'] ."' AND id in $in"; // #14

            if ($nodes = $mysqli->query($sqlquery)) { // #15

                $count = $nodes->num_rows; // #16

                if ($count > 0){ // #17

                    $r['leaf'] = false; // #18
                    $r['items'] = array(); // #19

                    while ($item = $nodes->fetch_assoc()) {
                        $item['leaf'] = true; // #20
```

```
                        $r['items'][] = $item; // #21
                    }
                }
                $folder[] = $r; // #22
            }
        }
    }
    $resultdb->close(); // #23
}
```

在这部分的代码里，先执行前一部分代码中准备好的SQL查询语句，获取用户有相应访问权限的Menu表的ID字段（#6）。然后，连接所有的ID值，并放在括号内，作为后面in运算符的操作列表（#7、#8和#9）。因为要重用resultdb变量，所以需要及时释放它（#10）。

接下来，要获取Menu表中表示一个模块的所有记录（#11）；本例中，模块的parent_id应为null，同时，id必须存在于in变量表示的操作列表中，也就是说，我们只想获取登录用户有相应权限的模块。然后执行SQL语句（#12）。

对每一个模块（#13），选择Menu表中表示树形面板节点的记录（也就是Packt.model.menu.Item模型类的实例），因此，需要执行一个新的SQL语句（#15），根据用户具有相应访问权限（id in $in）的模块（parent_id）选择属于它的所有菜单项（#14）。

另外，还需要获取结果集返回的记录数（#16）。如果结果集返回的记录数大于零（意味着返回了记录#17），则认为这个模块不是一个叶子（Ext JS的node interface即节点接口的概念，#18），然后初始化items属性（#19）。每个节点可当成一个叶子（#20），并分配节点给它的父模块（#21）。

最后，要把模块加到返回给Ext JS的集合中（#22），然后关闭结果集（#23）。

现在，我们来实现服务器端代码的最后部分：

```
$mysqli->close(); // #24

echo json_encode(array( // #25
    "items" => $folder
));
```

关闭MySQL连接（#24），把信息编码成JSON格式，并按前面Packt.store.Menu代码中proxy reader指定的，包装给items属性（#25）。

如果树形面板树层次大于2，我们的服务器端代码就无法运行了。在本例，树形面板仅有2层：根节点以及一级子节点。若想创建多级菜单，那就要用递归算法从数据库里获取数据，实现起来就复杂多了。

运行服务器端代码，将正确返回如4.1.2节列出的JSON对象。

菜单数据库表完美适配Ext JS，Menu表是根据Ext JS所需设计，可方便获取数据。前面的服务器端代码同样适配Ext JS。如果我们有从零开始设计数据库的机会，就很可能有不同设计，那

么服务器端代码获取信息的方式也会不同。数据库和服务器端代码的实现并非问题所在。很不幸，Ext JS要求信息以特定格式返回给前端代码，那么我们就必须这么做。如果从数据库获取的信息不符合Ext JS的要求（如前述格式），那就只好去解析它，那么在把信息返回给Ext JS之前，服务器端就多了一个处理环节。

4.1.5　用折叠面板和树形面板创建菜单

回到Ext JS代码，实现动态菜单组件。首先，在app/view下创建一个新目录menu，并创建新文件Accordion.js：

```
Ext.define('Packt.view.menu.Accordion', {
    extend: 'Ext.panel.Panel',
    alias: 'widget.mainmenu',

    width: 300,
    layout: {
        type: 'accordion'
    },
    collapsible: false,
    hideCollapseTool: false,
    iconCls: 'sitemap',
    title: 'Menu'
});
```

这个类是一个面板，放置菜单项并使用折叠布局。这样用户就可以展开需要的模块。将`title`设为Menu（可以使用语言转换文件，在en.js、es.js和pt_BR.js里设置相应属性），并设置`iconCls`美化。

下一步，创建代表每个模块的树形面板。创建新类`Packt.view.menu.Item`和新文件Item.js，文件位于app/view/menu目录下：

```
Ext.define('Packt.view.menu.Item', {
    extend: 'Ext.tree.Panel',
    alias: 'widget.mainmenuitem',

    border: 0,
    autoScroll: true,
    rootVisible: false
});
```

`Packt.view.menu.Item`类是树形面板。菜单不需要边框，此外要隐藏根节点。我们并没有在此设置太多属性，其余属性将在控制器里动态设置。

4.1.6　在视见区替换中央区域容器

实现控制器之前，需要再创建一个组件：包含了通过菜单打开的所有界面的标签面板。创建`Packt.view.MainPanel`类，相应文件为MainPanel.js，位在app/view/目录下：

```
Ext.define('Packt.view.MainPanel', {
    extend: 'Ext.tab.Panel',
    alias: 'widget.mainpanel',

    activeTab: 0,

    items: [
        {
            xtype: 'panel',
            closable: false,
            iconCls: 'home',
            title: 'Home'
        }
    ]
});
```


这个类有一个默认的activeTab为0，是items属性配置里的唯一项。活动标签面板默认为空面板，不允许关闭，有一个主页图标，标题为Home（重申一次，只要你愿意，可以在语言转换文件里创建一个新属性，并使用它）。

接下来，在app/view/MyViewport.js文件里用这个标签面板替换中央区域容器。

```
{
    xtype: 'mainpanel',
    region: 'center'
}
```

现在，标签面板可以“收纳”菜单打开的组件了。

我们还要用动态菜单替换左侧容器：

```
{
    xtype: 'mainmenu',
    width: 185,
    collapsible: true,
    region: 'west'//,
    //style: 'background-color: #8FB488;'
}
```

我们注释或者移除style行。用折叠面板的xtype替代xtype属性项设置。接下来让我们回到控制器。

4.1.7 创建菜单控制器

我们实现了所有的视图、模型、存储器以及服务器端代码，只剩下控制器还没实现，所有的逻辑都在其中处理。让我们继续，在app/controller目录下创建Menu.js文件。

```
Ext.define('Packt.controller.Menu', {
    extend: 'Ext.app.Controller',

    models: [
        'menu.Root',
```

```
        'menu.Item'
    ],
    stores: [
        'Menu'
    ],
    views: [
        'menu.Accordion',
        'menu.Item'
    ],

    refs: [
        {
            ref: 'mainPanel',
            selector: 'mainpanel'
        }
    ],

    init: function(application) {
        this.control({
            "mainmenu": {
                render: this.onPanelRender
            },
            "mainmenuitem": {
                select: this.onTreepanelSelect,
                itemclick: this.onTreepanelItemClick
            }
        });
    }
});
```

我们添加了与Menu相关的models、stores和views。同时，添加了mainpanel的引用——中央区域的标签面板。

我们将监听3个事件。第一个是渲染动态菜单。用户一旦通过认证，视见区界面就会呈现出来。包容了动态菜单的折叠面板（xtype：mainmenu，MyViewport类的一个子项）也将被渲染。此外要监听树形面板节点（菜单选项）相关的事件：select和itemclick事件。用户选择了某个菜单项，我们将在标签面板以一个新标签页打开相应界面。监听itemclick是因为用户可能无意间关闭了相应界面并希望再次打开。如果菜单项已选择过，界面就不会再次打开[①]，所以需要监听select事件。

1. 渲染嵌套JSON为数据源的菜单（**hasMany**绑定）

创建一个方法，用服务器端返回信息创建动态菜单：

```
onPanelRender: function(abstractcomponent, options) {
    this.getMenuStore().load(function(records, op, success){ // #1

        // #2
        var menuPanel = Ext.ComponentQuery.query('mainmenu')[0];
```

① 如果关闭了相应界面且无法再次打开，就出问题了。——译者注

```
            Ext.each(records, function(root){ // #3

                var menu = Ext.create('Packt.view.menu.Item',{ // #4
                    title: translations[root.get('text')], // #5
                    iconCls: root.get('iconCls') // #6
                });
                Ext.each(root.items(), function(itens){ // #7

                    Ext.each(itens.data.items, function(item){

                        menu.getRootNode().appendChild({ // #8
                            text: translations[item.get('text')],
                            leaf: true,
                            iconCls: item.get('iconCls'),
                            id: item.get('id'),
                            className: item.get('className')
                        });
                    });
                });

                menuPanel.add(menu); // #9
            });
        });
    }
```


第一件事就是加载MenuStore，它负责加载服务器端返回的嵌套JSON数据（#1）。存储器加载完成（这也是在加载回调函数里创建动态菜单的原因），需要获取Packt.view.menu.Accordion类的引用。我们尚未获取控制器的引用（在load回调函数里），通过Ext.componentQuery.query（#2）获取该引用（xtype：mainmenu）。这个查询方法返回一个引用数组，我们只需要第一个（也仅有一个满足条件的组件引用）。另外，在方法开头给this设置一个引用（var me = this;），然后在回调函数里，用me变量引用控制器。

对Store返回的每条记录（#3），创建一个树形面板（Packt.view.menu.Item，#4）表示每个模块，并设置title（从语音转换文件中获取，#5）和iconCls（#6）属性项。



Ext.create和Ext.widget

当本书第一次讨论类实例化的几种方式时（第2章），简述了可以使用Ext.create并传入完整类名作为参数，或者可以使用Ext.widget并传入类别名作为参数（还有其他方法，但这两种是最常用的）。这只是个人偏好而已，你可以使用我们在第2章里提到的各种方法。

接下来，对每个Packt.model.menu.Item模型实例（#7），获取树形面板的根节点，然后追加一个新节点（#8），同时设置text、leaf和iconCls等NodeInterface类的配置项。id和className是我们添加的配置项。以后访问这些配置项时，需要从raw属性项中获取。

最后，添加树形面板至折叠菜单。这个方法运行之后，动态菜单就完成了渲染。

2. 动态打开菜单项

菜单渲染之后，用户可以选择一个菜单项。这个方法的处理逻辑是：当用户选择了菜单项，我们需要判定屏幕上是否创建了相应的标签面板。如果是，则不再创建，只须选择并激活该面板；如果否，就需要创建它。控制器相应代码如下：

```
onTreepanelSelect: function(selModel, record, index, options) {
    var mainPanel = this.getMainPanel(); // #1

    var newTab = mainPanel.items.findBy( // #2
    function (tab){
        return tab.title === record.get('text'); // #3
    });

    if (!newTab){ // #4
        newTab = mainPanel.add({            // #5
            xtype: record.raw.className,    // #6
            closable: true,                 // #7
            iconCls: record.get('iconCls'), // #8
            title: record.get('text')       // #9
        });
    }
    mainPanel.setActiveTab(newTab); // #10
}
```

首先，获取标签面板引用（#1）。然后，比较标签`title`跟选择节点的`text`是否相同（#3），从而验证选择菜单项对应的标签界面是否已创建（#2）。

如果不存在新标签（#4），则添加到标签面板，作为实例传给`add`方法（#5）。通过节点`className`获取添加组件的`xtype`（#6），标签可关闭（#7），跟对应节点有相同的`iconCls`（#8）和`title`（#9，菜单项）。

之后，设置对应标签为活动标签。这里，界面已渲染，只需设置用户所选界面为活动标签即可（#10）。

此时如果用户关闭当前所选菜单项（对应标签），然后再次点击该菜单项，将不会有任何反应。用户得选择另一个菜单项，再回头选择该菜单项，这才有效。为避免这种情况，我们需要监听`itemclick`事件：

```
onTreepanelItemClick: function(view, record, item, index, event, options){
    this.onTreepanelSelect(view, record, index, options);
},
```

这个方法将调用`onTreepanelSelect`方法，不需重复代码。

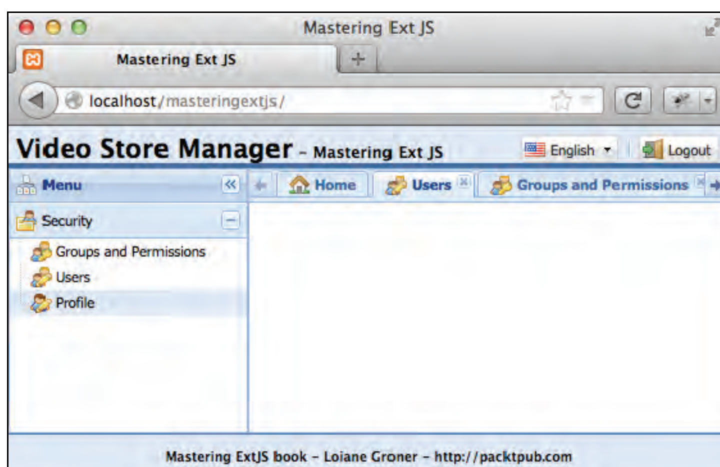
4.1.8　改动app.js

最后的一步是在app.js中添加新控制器：

```
controllers: [
    'Login',
    'TranslationManager',
    'Menu'
]
```

本书创建的每个控制器，都需要添加到app.js中的controllers配置项，下一次申明新控制器时别忘了这点。

执行代码并打开一个Menu（菜单）项，系统会打开一个空面板，因为还没有实现界面（下一章内容）。但我们已经可以看到动态菜单如下图所示：



动态菜单功能完成了。

4.2 小结

本章我们学习了通过折叠面板和树形面板为每个模块创建动态菜单的方法，了解了服务器端的动态逻辑处理，以及如何在Ext JS端处理服务器端的返回信息来创建动态菜单。最后，我们还学习了如何通过编程打开菜单项，并在中央区域标签面板里显示相应界面。

下一章我们将实现界面列表、创建及修改用户，并为用户分配用户组。

第 5 章 用户鉴权与安全

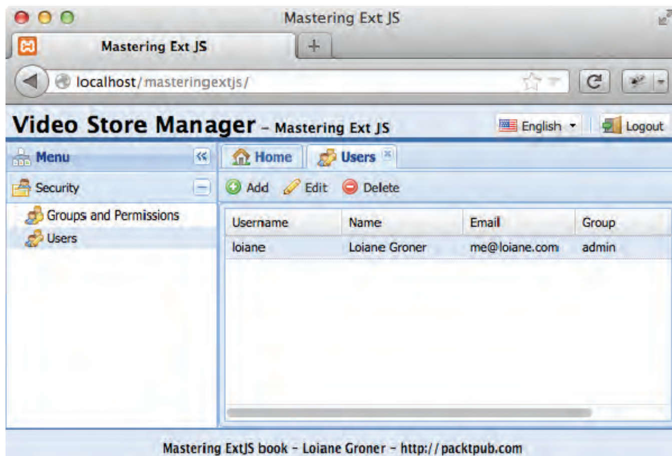
前面几章我们完成了登录和注销功能、会话监控，并基于用户权限实现了动态菜单。然而，目前所有的用户、用户组以及权限都是直接添加到数据库中。我们不可能每次给新用户授权或改变用户权限时都这么做，因此需要提供一个界面让我们可以创建新用户，授权或改变权限。本章内容如下：

- 列出所有系统用户；
- 创建、编辑和删除用户；
- 文件上传预览（用户图片）。

5.1 用户管理

用户管理是本章第一个要开发的模块。通过用户管理模块，我们可以查看所有注册用户、添加新用户、编辑以及删除当前用户。

用户点击Users Menu（用户菜单）项，将打开一个用户列表操作界面，如下图所示：



当用户点击Add（添加）或Edit（编辑）按钮，系统会提供一个窗体给用户用以创建新用户或编辑当前用户信息（基于网格面板上所选记录）。编辑窗体如下图所示：

创建或编辑用户的功能包括：编辑User Information（用户信息），如Name（姓名）、Username（用户名）等。我们还可以上传用户图片。还有一个额外的功能：利用HTML5 API新特性，我们可以在用户从电脑选择图片但未上传至服务器之前预览图片（Picture）。

继续前进！

5.2　列出所有用户：简单的网格面板

我们需要实现如本章第一张图所示那样的界面——一个简单的网格面板。实现思路如下。

- 模型——表示存储在用户表里的信息。
- 存储器——通过代理获取服务器端信息。
- 网格面板——表示视图。
- 控制器——监听网格面板渲染完成与否，完成后加载用户信息。

5.2.1　用户模型

首先创建一个模型，表示User（用户）表。在app/model/security目录下创建一个新文件User.js。这个模型表示User表里password字段之外的所有字段，密码是非常私密的信息，不能显示给其他用户，包括管理员。用户模型代码如下所示：

```
Ext.define('Packt.model.security.User', {
    extend: 'Ext.data.Model',

    idProperty: 'id',

    fields: [
        { name: 'id' },
```

```
            { name: 'name' },
            { name: 'userName' },
            { name: 'email' },
            { name: 'picture' },
            { name: 'Group_id' }
        ]
    });
```

如前面我们提到的，除了`password`字段，`User`表的其他字段都映射到这个模型中。

5.2.2　用户存储器

现在我们有了代表`User`表记录的模型，接下来要创建存储器，并设置一个代理加载数据库用户信息集合。在app/store/security目录下创建新文件Users.js（“Users”为复数形式，因为要处理用户信息集合）。请注意，我们在model和store目录下都创建了一个security新目录。在views和controller里也将做类似处理，这种遵循一定模式，在每个包里都创建文件的方式让代码维护变得很容易。

用户存储器代码如下：

```
Ext.define('Packt.store.security.Users', {
    extend: 'Ext.data.Store',

    requires: [
        'Packt.model.security.User' // #1
    ],

    model: 'Packt.model.security.User', // #2

    proxy: {
        type: 'ajax',
        url: 'php/security/users.php',

        reader: {
            type: 'json',
            root: 'data'
        }
    }
});
```

存储器表示用户模型的集合（#2），这个显式声明很重要：`the Packt.model.security.User`类需在存储器加载前就完成加载（#1）。使用Ajax代理从服务器端获取JSON格式的用户集合数据，用户集合数据包装在`data`属性内，如以下格式：

```
{
    "success": true,
    "data": [{
        "id": "1",
        "name": "Loiane Groner",
```

```
            "userName": "loiane",
            "email": "me@loiane.com",
            "picture": "FE03.png",
            "Group_id": "1"
        }]
    }
```

本章没有列出服务器端代码。你可以从本书支持网站获取完整代码：https://github.com/loiane/masteringextjs。

5.2.3 用户网格面板

接下来创建视图，管理系统用户。编码之前需要记住：当实现编辑用户组界面时，需要显示隶属于用户组的所有用户。这时候需要一个用户列表，也就是说，需要创建一个组件列出用户（这里指所有系统用户），后面还要重用它。因此，我们创建的组件只包含用户列表，不包括Add（添加）、Edit（编辑）或Delete（删除）按钮。我们将添加一个包含这些按钮的工具栏以及用户网格面板到另一组件中。

我们来创建网格面板。在app/view/security目录下创建UsersList.js新文件，在文件里创建新类`Packt.view.security.UsersList`：

```
Ext.define('Packt.view.security.UsersList', {
    extend: 'Ext.grid.Panel',
    alias: 'widget.userslist',

    frame: true,
    store: Ext.create('Packt.store.security.Users'), // #1

    columns: [
        {
            width: 150,
            dataIndex: 'userName',
            text: 'Username'
        },
        {
            width: 200,
            dataIndex: 'name',
            flex: 1,
            text: 'Name'
        },
        {
            width: 250,
            dataIndex: 'email',
            text: 'Email'
        },
        {
            width: 150,
```

```
                dataIndex: 'Group_id',
                text: 'Group',
                renderer: function(value, metaData, record ){ // #2
                    var groupsStore = Ext.getStore('groups');
                    var group = groupsStore.findRecord('id', value);
                    return group != null ? group.get('name') : value;
                }
            }
        ]
    });
```

上面的代码实现了一个简单网格面板，并显示从服务器端获取的信息。关于网格面板，有三个重点需要进一步讨论。第一个是每列的dataIndex。注意dataIndex必须匹配模型里的字段名。另外，我们要特别注意Ext JS是大小写敏感的，所以字段名group_id跟Group_id是两码事。

第二个重点是store的声明（#1）。请注意我们并未采用Ext JS模型–视图–控制器（MVC）应用开发中的常用做法，只是简单地声明store为security.Users。我们显式实例化了一个新的store实例。为什么这样做呢？这有何不同吗？是有很大不同！MVC架构通过存储管理器获取存储器实例的引用，而存储管理器决定了只有一个引用， 因为我们通过存储器ID获取引用。只有一个存储器引用贯穿整个应用（存储器在不同的组件和界面之间共享），这意味着一旦改变存储器，该存储器的其他所有实例也将改变（添加新记录、修改或删除等）。有时候我们并不希望这样，而是希望根据不同目的有不同的存储器实例。因此，在这里实现了一个独立的存储器引用（在实现用户组管理模块时会阐述原因，届时我们会重用用户列表加载指定用户组的用户）。我们需对上述内容给予关注。

第三个重点是Group_id列的renderer（渲染器）函数（#2）。当从服务器端加载存储器时，用户存储器的每个模型实例只包含Group_id列名而没有组名。我们不可能仅仅显示数字给用户，因为这对用户而言是没有意义的，我们还得显示组名。但我们现在并没有组名信息，需要考虑从何处获得它。事实上，可以从稍后要创建的，包含了所有用户组信息的groups存储器里获得组名 。因此，首先传入storeId参数给存储管理器并获取存储器；然后获取Group_id值对应的用户组模型实例；接下来，判断存储器里是否有匹配id的用户模型实例：如果有，就显示组名，否则就显示Group_id。我们可在一对一、一对多或多对多等数据库表关联关系中应用这种方法。



> 当从另一存储器里获取值时，须确保值存在于存储器中。这意味着存储器必须包含所有值，并且不能对其进行分页或过滤。需找到所需ID之后才能进行数据分页。另一种办法是实现一个具有非持久化字段的模型，只用于显示，这种方式下就不用在别的存储器中搜寻值了。当然，这也意味着需从服务器端获取更多信息。我们得评估不同情形从而的确定最佳方案。

现在我们有了用户网格面板，还需要一个组件，用来包含用户网格面板以及带有添加、编辑

和删除按钮的工具栏。面板是提供停驻项的最简单组件，因此，创建一个扩展自Ext.panel.Panel的新类Packt.view.security.Users。在app/view/security目录下创建一个Users.js新文件：

```
Ext.define('Packt.view.security.Users', {
    extend: 'Ext.panel.Panel',
    alias: 'widget.users',

    requires: [
        'Packt.view.security.UsersList' // #1
    ],

    layout: {
        type: 'fit' // #2
    },

    items: [
        {
            xtype: 'userslist' // #3
        }
    ]
});
```

面板里只渲染了一个用户网格面板组件（#3）。由于用了xtype实例化户网格面板，因此需要在requires声明里添加它（#1）。只有一个组件并且希望它能占满面板所有剩余空间时，就用fit布局（#2）。

下一步要添加停驻于顶部带有添加、编辑和删除按钮的工具栏，在Packt.view.security.Users类的dockedItems里声明它：

```
dockedItems: [
    {
        xtype: 'toolbar',
        flex: 1,
        dock: 'top',
        items: [
            {
                xtype: 'button',
                text: 'Add',
                itemId: 'add',
                iconCls: 'add'
            },
            {
                xtype: 'button',
                text: 'Edit',
                itemId: 'edit',
                iconCls: 'edit'
            },
            {
                xtype: 'button',
                text: 'Delete',
                itemId: 'delete',
```

```
                iconCls: 'delete'
            }
        ]
    }
]
```

为了后续能在控制器里识别每个按钮点击事件，我们为每个按钮设置了itemId项。现在我们的组件完成了。

5.2.4 用户控制器

到最后一步了，我们需要一个界面以加载系统的所有用户，当用户网格面板准备就绪并渲染之后，这个界面将加载存储器。这部分我们需要在控制器里实现。接下来，创建控制器：在app/controller/security目录下创建新文件Users.js，创建Packt.controller.security.Users类：

```
Ext.define('Packt.controller.security.Users', {
    extend: 'Ext.app.Controller',

    views: [
        'security.Users' // #1
    ],

    init: function(application) {

        this.control({
            "userslist": { // #2
                render: this.onRender
            }
        });
    },

    onRender: function(component, options) { // #3
        component.getStore().load();         // #4
    }

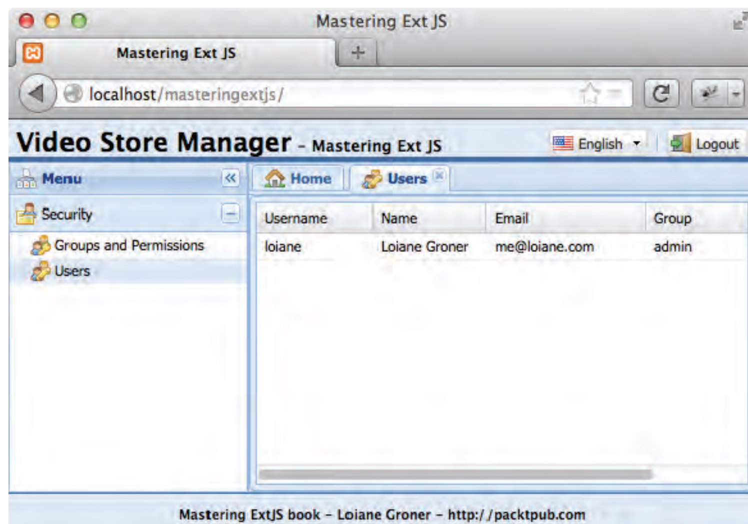

});
```

首先，在views属性里声明用户网格面板视图（#1）。然后，开始监听用户网格面板的渲染事件（#2）。当render（渲染）事件触发，执行onRender方法（#3），在其中实现用户存储器的加载。再次强调一下：因为用户网格面板视图负责实例化存储器，所以我们不能在stores属性里声明用户存储器，因此控制器不存在直接获取存储器的方法。我们通过获取用户网格面板引用，进一步获取用户存储器引用，然后加载用户存储器；这正是onRender方法所做的。

接下来，把控制器添加到app.js文件中，修改Menu数据库表（修改用户记录）的className列值为Users，如下图所示：

id	text	iconCls	parent_id	className
1	menu1	menu_admin	NULL	NULL
2	menu11	menu_groups	1	panel
3	menu12	menu_users	1	Users

运行程序，可以看到所有系统用户的列表：



5.3　添加和编辑用户

我们实现了列出所有系统用户的功能，接下来要实现添加和编辑功能。但在控制器中添加新的事件监听器之前，需要新建视图，作为编辑或添加用户的界面。

5.3.1　创建编辑视图：窗体里的表单

新视图是个弹出窗体，包含用户信息表单、工具栏，以及底部的两个按钮：Cancel按钮及Save按钮。窗体很像第2章里实现的登录窗体，但将实现新的功能，如表单上传、用HTML5新特性实现文件预览等。

创建扩展自Window类的新类Packt.view.security.Profile：

```
Ext.define('Packt.view.security.Profile', {
    extend: 'Ext.window.Window',
    alias: 'widget.profile',
```

```
    height: 260,
    width: 550,

    requires: ['Packt.util.Util'],

    layout: {
        type: 'fit'
    },
    title: 'User',

    items: [
        {
            xtype: 'form',
            bodyPadding: 5,
            layout: {
                type: 'hbox', // #1
                align: 'stretch'
            },
            items: [

            ]
        }
    ]
});
```

其中有两点需要说明，第一是不使用autoShow属性。这样我们就可以在创建好窗体并设置一些属性后，通过show方法调用来呈现它。

第二点是表单的布局。我们没有使用表单默认布局（anchor布局），而是采用hbox布局（#1），这样就可以水平组织表单里的组件项。我们还希望组件占满垂直剩余空间，因此将align设置为stretch，这样就不用为每个组件设置高度。

现在来给表单添加第一个组件。回头看看本章开头的图示，可发现我们打算用两个fieldset类型组织表单组件。因此，首先实现的组件就是fieldset，用以组织用户信息。

```
{
    xtype: 'fieldset',
    flex: 2,
    title: 'User Information',
    defaults: {
        afterLabelTextTpl: Packt.util.Util.required, // #1
        anchor: '100%',
        xtype: 'textfield',
        allowBlank: false,
        labelWidth: 60
    },
    items: [
    ]
},
```

给所有必填组件字段打上红星标记（#1）。afterLabelTextTpl属性是个字符串或

XTemplate配置，放在标签文本后面的标记位上。像大多数必填字段一样，需要声明allowBlank属性为false；同时声明xtype属性默认设置为textfield。后续如果某些字段不需要默认设置，我们再取其他值代替。接下来，声明fieldset中items属性设置项里的组件字段：

```
{
    xtype: 'hiddenfield',
    fieldLabel: 'Label',
    name: 'id'
},
{
    fieldLabel: 'Username',
    name: 'userName'
},
{
    fieldLabel: 'Name',
    maxLength: 100,
    name: 'name'
},
{
    fieldLabel: 'Email',
    maxLength: 100,
    name: 'email'
},
{
    xtype: 'combobox',
    fieldLabel: 'Group',
    name: 'Group_id', // #1
    displayField: 'name',  //#2
    valueField: 'id', // #3
    queryMode: 'local', // #4
    store: 'security.Groups' // #5
},
{
    xtype: 'filefield',
    fieldLabel: 'Picture',
    name: 'picture',
    allowBlank: true, // #6
    afterLabelTextTpl: '' // #7
}
```

隐藏id字段，因为我们仅在内部使用且不希望将其呈现给用户，userName、name和email为简单文本字段。

接着实现组合框。将用户模型Group_id字段（#1）映射至组合框；当已有的用户加载进表单待编辑时，获取Group_id，匹配用户组的id（已是组合框的值，#3），组合框显示用户组名称（#2）。我们使用了groups存储器（#5，与用户网格面板渲染函数里使用的存储器实例是同一个），组合框只加载一次groups存储器（#4）。这样一来，打开组合框时，不用每次都加载服务器端数据。

现在实现文件上传字段。由于该字段非必填项（#6），就不需要显示红星标记（#7），这里，我们覆写了默认设置。

以上实现了第一个fieldset组件，显示在表单左边。下一步，实现另一个位于表单右边并包含一张图片的fieldset组件。

```
{
    xtype: 'fieldset',
    title: 'Picture',
    width: 170, // #1
    items: [
        {
            xtype: 'image', // #2
            height: 150,
            width: 150,
            src: ''         // #3
        }
    ]
}
```

这个fieldset组件声明了固定宽度（#1），其宽度将固定。表单使用hbox布局，而前面实现的fieldset组件设置了flex属性，所以它将占满剩余水平空间。

在Picture fieldset组件里，我们使用了一个Ext.Image组件。Ext.Image（#2）类用来创建并渲染图片。它会用特定的src（里面是代码片段）在文档对象模型（Document Object Model，DOM）中创建一个<image>标签（#3）。src属性目前为空。当加载已有的用户并编辑时，将在Image组件中显示用户图片（如果有的话）。同样，如果用户上传新图片，也将在其中预览。

最后一步是声明一个带有保存和取消按钮的底部工具栏。

```
dockedItems: [
    {
        xtype: 'toolbar',
        flex: 1,
        dock: 'bottom',
        ui: 'footer',
        layout: {
            pack: 'end', // #1
            type: 'hbox'
        },
        items: [
            {
                xtype: 'button',
                text: 'Cancel',
                itemId: 'cancel',
                iconCls: 'cancel'
            },
            {
                xtype: 'button',
                text: 'Save',
```

```
                    itemId: 'save',
                    iconCls: 'save'
                }
            ]
        }
    ]
```

我们希望按钮排列在工具栏右侧，因此使用了hbox布局并通过pack属性配置项把按钮放至工具栏末端（#1）。现在，编辑/添加窗体已准备就绪，但在实现添加和编辑监听器之前，仍有一些细节需要考虑。

5.3.2 用户组模型

在用户网格面板和用户组组合框的实现里，我们声明了groups存储器，用来从数据库加载所有的用户组。现在我们就来实现它，第一步要创建表示Groups数据库表单条记录的模型，Packt.model.security.Group模型代码如下：

```
Ext.define('Packt.model.security.Group', {
    extend: 'Ext.data.Model',

    idProperty: 'id',

    fields: [
        { name: 'id' },
        { name: 'name' }
    ]
});
```

Groups表很简单，只包含了两列（id和name），因此group模型也就简单地包含这两个相应字段。

5.3.3 用户组集模型

创建完group（用户组）模型后，接下来创建groups（用户组集）模型。



要始终养成这种命名习惯：模型（model）名称用实体名称的单数形式，存储器（store）名称用模型/实体名称的复数形式。

创建新的存储器Packt.store.security.Groups：

```
Ext.define('Packt.store.security.Groups', {
    extend: 'Ext.data.Store',

    requires: [
        'Packt.model.security.Group'
```

```
    ],

    model: 'Packt.model.security.Group',

    storeId: 'groups', //#1

    proxy: {
        type: 'ajax',
        url: 'php/security/group.php',

        reader: {
            type: 'json',
            root: 'data'
        }
    }
});
```

这个存储器跟我们已创建的存储器很相似。唯一要注意的是，这里我们声明了一个storeId属性。当我们执行用户网格面板Group_id列的渲染函数代码（Ext.getStore('groups')）时，声明storeId属性对于通过存储管理器获取存储器是个好方式。storeId属性在应用程序范围内给存储器分配唯一的ID值。

同样，Groups存储器跟其他存储器差不多，通过JSON格式数据的data属性取回服务器端用户组集信息：

```
{
    "success": true,
    "data": [{
        "id": "1",
        "name": "admin"
    }]
}
```

现在，相关视图、模型和存储器都已创建。接下来我们实现控制器中所需的监听事件。

5.3.4　控制器：监听Add按钮事件

当用户点击（Add）按钮时，将呈现编辑用户窗体（Packt.view.security.Profile类）。

第一步，在用户控制器this.control声明中添加事件监听器：

```
"users button#add": {
    click: this.onButtonClickAdd
}
```

我们要找的添加按钮在users组件（Packt.view.security.Users类）里，并且将按钮的itemId设置为add，所以选择器为users button#add。编辑和删除按钮的选择器也类似，只需更改相应的itemId即可。

当用户点击Add按钮时，控制器执行onButtonClickAdd方法：

```
onButtonClickAdd: function (button, e, options) {
    var win = Ext.create('Packt.view.security.Profile');
    win.setTitle('Add new User');
    win.show();
}
```

在这个方法中，我们创建了一个编辑窗体的实例，并将标题设置为Add new User（这里可以使用前面实现的多语言支持功能），最后将这个可以把新的用户信息发送给系统用户的窗体呈现出来。

5.3.5 控制器：监听Edit按钮事件

我们也可以编辑现有用户，所以需要监听编辑按钮点击事件：

```
"users button#edit": {
    click: this.onButtonClickEdit
}
```

可以看到，编辑按钮选择器与添加按钮非常类似，差异之处只在于itemId不同。

当用户点击Edit按钮时，控制器执行onButtonClickEdit方法：

```
onButtonClickEdit: function (button, e, options) {
    var grid= this.getUsersList(), // #1
    record = grid.getSelectionModel().getSelection();

    if(record[0]){ // #2
      var editWindow = Ext.create('Packt.view.security.Profile');

        editWindow.down('form').loadRecord(record[0]); // #3

        if (record[0].get('picture')) { //#4
            var img = editWindow.down('image');
            img.setSrc('resources/profileImages/' + record[0].get('picture'));
        }
        editWindow.setTitle(record[0].get('name')); // #5
        editWindow.show();
    }
}
```

至此，我们的代码还没有完成。想要编辑一个现有用户，还需要将所选用户的信息填入表单。首先，获取用户网格面板的引用，并获取所选用户记录（#1）。如果获取到所选用户记录（#2），就创建一个窗体实例；用所选用户记录信息加载表单（#3），设置窗体title属性为所选用户记录的name属性值（#5），之后显示该窗体。

还有一个细节：我们还打算显示用户图片，因此还要检查所选用户信息里是否包含图片（#4），

如果有图片，就获取Image组件的引用并设置source为用户图片。这样当上传一张新的用户图片时就会看到它，我们把图片保存在resources/profileImages目录中。也可以把图片直接保存在数据库里，但这会引起性能问题，因此还是保存在目录里。我们会在另一章学习怎样把文件保存到数据库。

5.3.6　控制器：保存用户信息

现在，用户可以打开编辑窗体添加或编辑用户信息了，接下来我们实现保存按钮。不管是添加新用户还是编辑已有的用户信息，我们都采用同样的保存方法。如果要用UPDATE或INSERT操作，可交给服务器端处理。

首先，为保存按钮添加监听器：

```
"profile button#save": {
    click: this.onButtonClickSave
}
```

记住，我们在Packt.view.security.Profile类里实现保存按钮的功能，因此选择器限定成profile（xtype属性）里itemId为save的按钮。

当用户点击Save按钮时，控制器执行onButtonClickSave方法。因为这个方法的代码比较长（指源代码行数），我们一步一步地实现它：

```
onButtonClickSave: function(button, e, options) {
    var win = button.up('window'), // #1
    formPanel = win.down('form'), // #2
    store = this.getUsersList().getStore();// #3

    if (formPanel.getForm().isValid()) { // #4
        formPanel.getForm().submit({// #5
            clientValidation: true,
            url: 'php/security/saveUser.php', // #6
            //成功和失败
    });
}
```

首先，获取窗体引用（#1），然后依次获取表单引用（#2）、用户网格面板存储器引用（#3）。我们打算在用户信息保存之后，关闭窗体，并重新加载用户网格面板存储器。

接下来验证表单是否合法（#4，用户根据客户端验证规则填充合法值），验证通过后提交表单（#5）至url（#6）对应页面。

可以通过存储器功能加载、创建、编辑和删除用户（如同后续阐述）。然而，我们现在采用另一种方式：直接通过提交方法发送数据给服务器端，因为还要上传一个文件给服务器端。

下一步要实现success和failure函数。我们先来实现success函数：

```
success: function(form, action) {
    var result = action.result; // #7
    if (result.success) {
        Packt.util.Alert.msg('Success!', 'User saved.'); // #8
        store.load();
        win.close();
    } else {
        Packt.util.Util.showErrorMsg(result.msg); // #9
    }
}
```

通常，我们先获取服务器端响应（#7）。如果成功，则显示给用户一个摘要通知（#8），然后关闭窗体并重新加载用户存储器，从服务器获取最新数据（请注意，如果我们使用存储器的sync方法从服务器端同步添加、编辑及删除的数据，就不需要重新加载用户存储器了）。如果服务器端出错（result.success为false），则显示服务器端错误信息（#9）。

下一步实现failure函数。因为我们实现了所有Ajax调用的默认failure函数处理代码，因此也对所有的表单提交failure函数实现默认处理：


```
failure: function(form, action) {
    switch (action.failureType) {
        case Ext.form.action.Action.CLIENT_INVALID:
            Ext.Msg.alert('Failure', 'Form fields may not be submitted with invalid
values');
            break;
        case Ext.form.action.Action.CONNECT_FAILURE:
            Ext.Msg.alert('Failure', 'Ajax communication failed');
            break;
        case Ext.form.action.Action.SERVER_INVALID:
            Ext.Msg.alert('Failure', action.result.msg);
   }
}
```

大体上，我们检查字段是否提交了合法值、是否有通信错误或者其他状况。

现在，我们的用户信息保存功能就完成了。

5.3.7 控制器：监听Cancel按钮

还剩一个取消按钮需要我们实现。在this.control设置里添加监听器：

```
"profile button#cancel": {
    click: this.onButtonClickCancel
}
```

当用户点击Cancel按钮时，控制器将执行onButtonClickCancel方法：

```
onButtonClickCancel: function(button, e, options) {
    button.up('window').close();
}
```

我们希望的实现很简单：如果用户希望取消对已有用户信息的修改或者新用户的添加，系统就关闭编辑窗体。由于取消按钮在编辑窗体里，因此先找到它然后调用close方法销毁窗体。

5.3.8　在上传之前预览文件

最后，为编辑窗体添加文件上传预览功能。这实现起来并不困难，却能让应用程序增色不少。

我们要做的是：当用户通过文件上传组件选择一个新文件时，使用HTML5的文件读取API读取文件。遗憾的是，并非所有的浏览器都支持这个API特性，只有以下版本的浏览器才支持：Chrome 6+、Firefox 4+、Safari 6+、Opera 12+、 Explorer 10+、iOS Safari 6+、Android 3+、Opera Mobile 12+。不过不用担心，我们可以首先判断浏览器是否支持API特性，如果不支持就不用这个特性，也就是说文件预览功能不会起作用。



若希望了解更多关于文件读取API的信息，请访问官方规范（http://www.w3.org/TR/file-upload/）；若希望了解更多HTML5的新特性，请访问http://www.html5rocks.com/。

当用户通过Ext JS文件上传组件选择了一个新文件时，change事件将被触发，因此要在控制器里监听该事件：

```
"profile filefield": {
    change: this.onFilefieldChange
}
```

接下来，我们需要实现onFilefieldChange方法：

```
onFilefieldChange: function(filefield, value, options) {
    var file = filefield.fileInputEl.dom.files[0]; // #1

    var picture = this.getUserPicture(); // #2

    if (typeof FileReader !== "undefined" && (/image/i).test(file.type)) {     // #3
        var reader = new FileReader();       // #4
        reader.onload = function(e){         // #5
            picture.setSrc(e.target.result); // #6
        };
        reader.readAsDataURL(file);          // #7
    } else if (!(/image/i).test(file.type)){ // #8
        Ext.Msg.alert('Warning', 'You can only upload image files!');
        filefield.reset();                   // #9
    }
}
```

首先，获取上传组件文件输入元素中的文件对象（#1，上传组件作为参数传入

onFilefieldChange方法）。之后，获取表单中Ext.Image组件的引用（#2），这样，我们就能够修改其source属性为预览文件。

我们还测试了浏览器是否支持文件读取API特性，以及用户所选文件是否为图片类型（#3）。如果两者都为true，则实例化FileReader（#4），并给它添加一个监听器（#5），这样，当FileReader实例完成读取文件，就可以将其内容设置给Ext.Image组件的source属性（#6）。当然，要触发onload事件，得靠FileReader实例读取文件内容（#7）。有一点至关重要：在上传服务器之前显示文件内容。如果用户保存了表单的变更，新的用户信息就会发送至服务器端（包括上传的文件），下一次打开编辑窗体时，就会显示图片了。



我们如何获取上传文件的完整路径呢？例如，Ext JS文件上传组件显示C:\fakepath\nameOfTheFile.jpg（但这并非真实路径），我们想获得其真实路径，如C:\Program Files\nameOfTheFile.jpg。答案是：利用JavaScript无法实现（而Ext JS是一个JavaScript框架）!

这种限制并非来自Ext JS；试试其他JavaScript框架/库，如jQuery，你会发现同样不可能实现，因为这是浏览器的安全约束。想像一下如果没有这种约束，可能就会有人开发出恶意JavaScript程序并在你浏览网页时获取你电脑上的信息。

有一个好消息：如果用户选择的文件不是图片文件（#8），我们将显示一条信息提示只有图片可以上传，并重置文件上传组件。但不幸的是，我们无法在浏览窗口过滤文件类型（打开窗口，你可以选择一个文件），这是一个让我们在Ext JS客户端而非服务器端进行验证的折衷方案。

如果FileReader实例不可用，就什么也不会发生。用户只能选择文件而无法预览。



可上传文件的大小取决于于Ext JS应用程序所部署的Web服务器软件对其的限制设定。例如，Apache限制为2 GB；**互联网信息服务**（Internet Information Services，IIS）默认设置为4 MB，但你可以将上限设置为2 GB；Apache Tomcat或其他Web服务器也类似。因此，上传文件大小的限制不在于Ext JS，而是Web服务器，所以你需要进行相应的设置。

5.4 删除用户

CRUD（Create Read Update Destroy）操作的最后一步是删除用户操作，我们在控制器里添加删除按钮监听器：

```
"users button#delete": {
```

```
    click: this.onButtonClickDelete
}
```

当用户点击删除（Delete）按钮时，控制器将执行onButtonClickDelete方法。

```
onButtonClickDelete: function (button, e, options) {
    var grid= this.getUsersList(),
    record = grid.getSelectionModel().getSelection(),
    store = grid.getStore();

    if (store.getCount() >= 2 && record[0]){

        // 在此删除逻辑

    } else if (store.getCount() == 1) {
        Ext.Msg.show({
            title:'Warning',
            msg: 'You cannot delete all the users from the application.',
            buttons: Ext.Msg.OK,
            icon: Ext.Msg.WARNING
        });
    }
}
```

实现思路是验证用户是否选择删除用户网格面板的一些记录（record[0]存在），并且系统存在两个及以上的用户（我们将只删除一个）。条件为真则删除用户，条件为假则意味着系统只有一个用户，我们不能删除这唯一存在的用户。

如果满足删除用户的条件，系统会提示是否确定要删除所选用户：

```
Ext.Msg.show({
     title:'Delete?',
     msg: 'Are you sure you want to delete?',
     buttons: Ext.Msg.YESNO,
     icon: Ext.Msg.QUESTION,
     fn: function (buttonId){
        if (buttonId == 'yes'){
            Ext.Ajax.request({
                url: 'php/security/deleteUser.php',
                params: {
                    id: record[0].get('id')
                },
                // 成功与失败
            });
        }
     }
});
```

如果确定删除操作，将发送所选用户的ID值到服务器端，并等待success或failure函数执行。我们总是期待操作能够成功，那就先来实现success函数：

```
success: function(conn, response, options, eOpts) {
    var result = Packt.util.Util.decodeJSON(conn.responseText);
    if (result.success) {
```

```
            Packt.util.Alert.msg('Success!', 'User deleted.');
            store.load();
        } else {
            Packt.util.Util.showErrorMsg(conn.responseText);
        }
    }
```

记住，服务器端执行数据库删除操作，多数情况下我们只是执行逻辑删除，也就是修改活动状态列标识（把用户改为不活动状态）。

我们需获取服务器端响应，如果成功，则显示一个通知并重新加载存储器。如果不成功，则像前几章的Ajax请求返回一样，显示一个错误信息。

failure函数代码如下：

```
failure: function(conn, response, options, eOpts) {
    Packt.util.Util.showErrorMsg(conn.responseText);
}
```


我们将服务器端返回的错误信息进行简单的显示。

接下来，我们需要做最后一个变更：修改Menu数据库表，显示本章创建的所有的组件和界面。修改className字段为users，这是包含用户网格面板、添加按钮、编辑按钮以及删除按钮的面板的xtype值。

Filter:　Edit:　Export:

id	text	iconCls	parent_id	className
1	menu1	menu_admin	NULL	NULL
2	menu11	menu_groups	1	panel
3	menu12	menu_users	1	users

现在我们重新加载应用程序并测试所有功能。

5.5　小结

本章讲解了如何创建、修改、删除以及列出所有系统用户。

我们还学习了一些有价值的理念，比如在整个Ext JS MVC应用程序中重用存储器，以及如何在控制器里获取其引用。我们还探讨了HTML5新特性——文件上传预览功能，这是又一个融合使用其他技术和Ext JS的例子。

下一章，我们将实现MySQL数据库表管理模块，其界面与MySQL workbench软件里的数据库表编辑界面非常相似。

第 6 章

MySQL数据库表管理

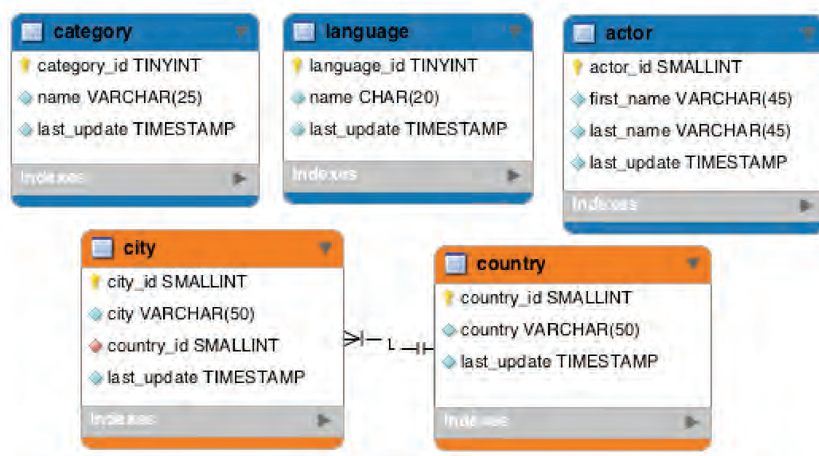
截至当前，我们实现了应用程序的基本功能。现在开始实现其核心功能，首先是MySQL数据库表管理。这是什么意思呢？每个应用程序都有无法直接关联到核心业务的信息，但这些信息又被核心业务以某种方式使用。比如，分类类别、用户语言、所在城市以及国家，这些信息都独立于核心业务存在并为核心业务所用。我们把这些信息组织为独立的MySQL数据库表（因为我们使用MySQL数据库服务器），在其上可以执行面向MySQL数据库表的各种操作。

本章主要包括以下内容：

- 新建模块Static Data（静态数据）；
- 列出MySQL数据库表的所有信息；
- 创建新的表记录；
- 即席搜索表；
- 过滤信息；
- 编辑、删除记录；
- 创建在所有表中重用的抽象组件。

6.1 呈现数据库表

如果我们打开并分析`Sakila`数据库自带的库表ER图（Entity Relationship，实体关系图），将看到下列数据库表：



这些表可以独立于其他表存在，我们将在本章使用它们。

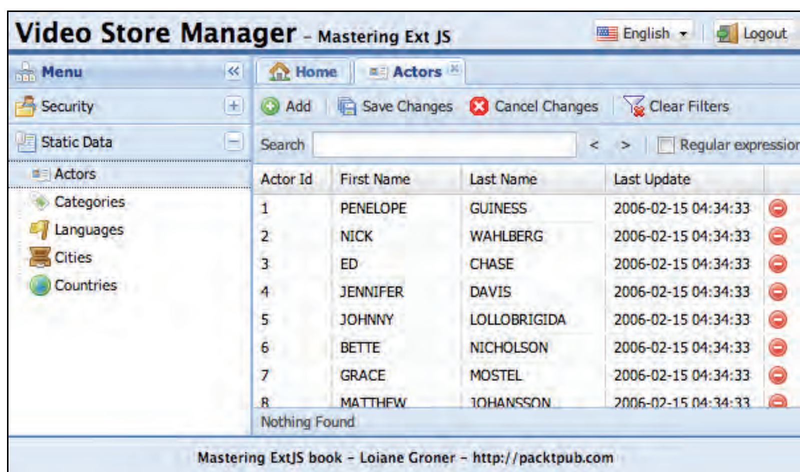
当我们在MySQL Workbench中打开SQL Editor（SQL编辑器），可以选中一个表；右击并选择Edit Table Data（编辑表数据）。按此操作，将打开一个新的标签页，如下图所示：

actor

Filter:　Edit:　Export:

actor_id	first_name	last_name	last_update
1	PENELOPE	GUINESS	2006-02-15 04:34:33
2	NICK	WAHLBERG	2006-02-15 04:34:33
3	ED	CHASE	2006-02-15 04:34:33
4	JENNIFER	DAVIS	2006-02-15 04:34:33
5	JOHNNY	LOLLOBRIGIDA	2006-02-15 04:34:33
6	BETTE	NICHOLSON	2006-02-15 04:34:33
7	GRACE	MOSTEL	2006-02-15 04:34:33
8	MATTHEW	JOHANSSON	2006-02-15 04:34:33
9	JOE	SWANK	2006-02-15 04:34:33
10	CHRISTIAN	GABLE	2006-02-15 04:34:33
11	ZERO	CAGE	2006-02-15 04:34:33
12	KARL	BERRY	2006-02-15 04:34:33
13	UMA	WOOD	2006-02-15 04:34:33

这个表是`actor`表（演员表）。我们打算对每个所选表都实现与这相似的界面：`Actors`、`Categories`、`Languages`、`Cities`和`Countries`（演员、类别、用户语言、所在城市以及国家等各表），如下图所示（这也是本章最终要实现的效果）：



本章的目标是以最少的代码实现这五张表的界面。也就是说我们希望创建最通用的代码，方便以后代码维护和功能增强，而且以通用功能为基础，创建新的界面也将变得更容易。

让我们继续。

6.2　创建模型

一般情况下我们先创建模型。首先列出涉及的数据库表及其字段。

- **Actor**　actor_id、first_name、last_name、last_update
- **Category**　category_id、name、last_update
- **Language**　language_id、name、last_update
- **City**　city_id、city、country_id、last_update
- **Country**　country_id、country、last_update

完全可以为每个实体都创建一个模型，但是我们希望尽可能重用代码。再看看列出的表和字段，注意所有表都有个相同的字段：last_update。

所有表都有同一列last_update，因此我们可以创建一个模型超类，然后再创建各表对应的特定模型（扩展自模型超类，不需要再声明last_update列）。这样不是挺好吗?

6.2.1　抽象模型

面向对象编程（Object Oriented Programming，OOP）里有个继承的概念，是一种重用已有对象代码的方式。Ext JS使用OOP方法，因此我们可在Ext JS应用程序里使用这个概念。如果回头看看已实现的代码，会发现我们已经在大部分类里应用了继承的概念（util包除外），但我们还

只是创建Ext JS类的继承类。现在，我们来创建自己的超类。

既然我们需要的模型都有last_update字段（仔细观察会发现所有的sakila数据库表都有这个字段），接下来就创建包含这个字段的模型超类。在app/model/sakila目录下创建新文件Sakila.js：

```
Ext.define('Packt.model.sakila.Sakila', {
    extend: 'Ext.data.Model',

    fields: [
        {
          name: 'last_update',
          type: 'date',
          dateFormat: 'Y-m-j H:i:s'
        }
    ]
});
```

后面我们将学习怎样用Sencha命令行工具构建产品，它使用了YUI Compressor（YUI压缩工具）压缩代码。因为这个原因，我们不能用“abstract”这个单词作为包名。虽然将Packt.model.abstract.Sakila作为类名非常合适，但我们不能这样使用。

这个模型只有一个last_update字段。在表里，last_update字段为timestamp（时间戳）类型，因此，这个字段是日期型，并且应用了Y-m-j H:i:s日期格式（year-month-day hour:minutes:seconds，年–月–日 时 : 分 : 秒），跟数据库里的格式一样（如：2006-02-15 04:34:33）。

接下来我们创建表示每张表的模型，不需要再次声明last_update了。

6.2.2 特定模型

现在，我们创建表示每张表的所有模型。我们先来实现Actor模型。在app/model/staticData目录下创建新文件Actor.js，并创建新类Packt.model.staticData.Actor：

```
Ext.define('Packt.model.staticData.Actor', {
    extend: 'Packt.model.sakila.Sakila', // #1

    idProperty: 'actor_id', // #2

    fields: [
        { name: 'actor_id' },
        { name: 'first_name'},
        { name: 'last_name'}
    ]
});
```

与前面创建的模型类相比，此处有两个不同地方。扩展类（#1）不是Ext.data.Model，而是Packt.model.sakila.Sakila。Sakila模型类已扩展自Ext.data.Model类，Packt.model.staticData.Actor类扩展自Sakila模型类。也就是说Actor模型类继承Sakila模型

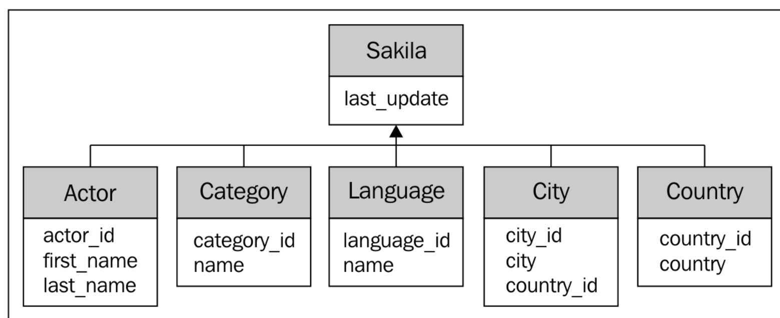
类，而Sakila模型类继承Ext.data.Model类。这样我们的Actor模型类就继承了所有Ext.data.Model类特性以及Sakila模型类的last_update字段。

另一个不同地方是idProperty（#2）。默认情况下，idProperty属性值为id。这样当我们声明一个带有名为id字段的模型时，Ext JS会自动识别这是模型的唯一字段（Unique Fields）。当idProperty属性值不为id时，我们需要显式指明idProperty属性值。由于所有的Sakila数据库表都没有名为id的唯一字段，idProperty属性值将设置为“实体名称+_id”格式。我们需要在所有特定模型中显式声明idProperty属性。

现在，对其他模型类进行类似的操作，创建4个模型类：

- Packt.model.staticData.Category
- Packt.model.staticData.Language
- Packt.model.staticData.City
- Packt.model.staticData.Country

最终，app/model/staticData包里有5个模型类。如果用UML类图来表达模型类，将如下图所示：



Actor、Category、Language、City和Country模型类都扩展自Sakila模型类，Sakila模型类扩展自Ext.data.Model模型类。

6.3　创建存储器

下一步我们创建每个模型对应的存储器。正如我们实现模型的方式，我们同样将创建一个通用存储器。虽然这个通用配置不在存储器中，而是在代理属性配置项里，但通过存储器超类能够监听所有静态数据存储器的通用事件。

我们将创建Packt.store.staticData.Abstract新存储器。

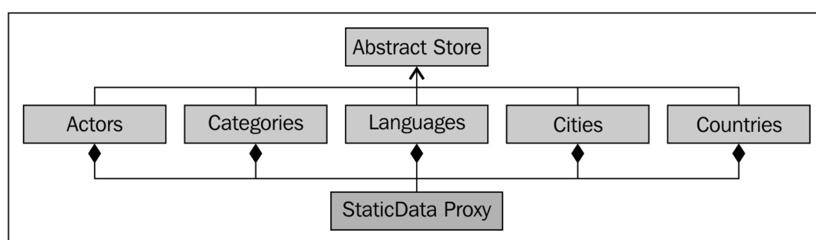
由于每个模型对应一个存储器，所以我们需创建以下存储器：

- Packt.store.staticData.Actors

- Packt.store.staticData.Categories
- Packt.store.staticData.Languages
- Packt.store.staticData.Cities
- Packt.store.staticData.Countries

最重要的配置是在代理属性配置项里声明的url、reader和writer等内容。其中一些配置对staticData包中所有存储器来说都是一样的。因此，我们创建一个超级代理类Packt.proxy.StaticData，从而尽可能重用代码。

最后，创建上述类，其UML类图表示如下：



所有的存储器类都扩展自Abstract存储器类，并在属性设置里都有StaticData代理配置项。

6

现在，我们理清了思路，下面就动手吧！

6.3.1 抽象存储器

我们首先创建Packt.store.staticData.Abstract类。我们并未在其中声明太多配置项，创建这个类唯一的目的就是声明一个store.staticData包中所有存储器都有的storeId属性：

```
Ext.define('Packt.store.staticData.Abstract', {
    extend: 'Ext.data.Store',

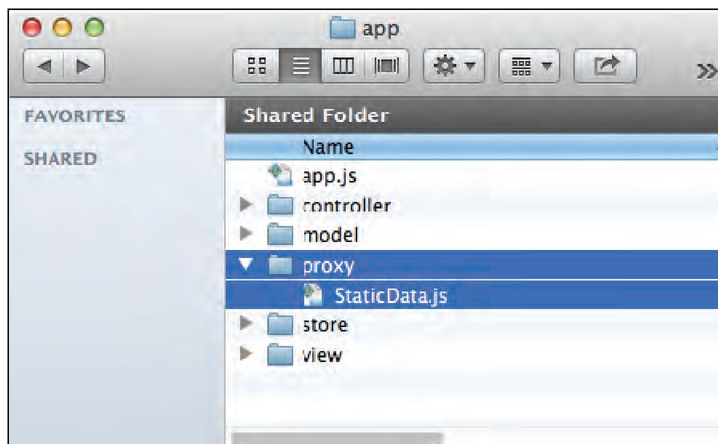
    storeId: 'staticDataAbstract'
});
```

所有的特定存储器都扩展自这个存储器。创建一个这样的存储器超类感觉没有多大意义。但是，我们并不知道将来维护时还得添加多少通用的存储器配置项。另一个这么做的理由是，在控制器里我们也能监听存储器事件（Ext JS 4.2及以上版本可用）。如果我们打算监听一系列存储器的同一事件并执行同一方法，有了存储器超类就能够节省很多行代码。

6.3.2 抽象代理类

接下来创建代理超类。我们将在其中实现staticData包里各个存储器的所有通用配置项。

我们要一步步完成。首先，在app目录下创建子目录proxy。然后，在proxy里创建StaticData.js新文件，如下图所示：



很显然，类名应该是Packt.proxy.StaticData。先来声明类的基本框架：

```
Ext.define('Packt.proxy.StaticData', {
    extend: 'Ext.data.proxy.Ajax',
    alias: 'proxy.staticdataproxy', // #1

    type: 'ajax' // #2
});
```

分配别名给代理类（#1）。给扩展自Ext JS组件类的所有类（如网格面板、树形面板、表单面板，等等）分配别名时，使用widget.前缀，后面加上我们分配给它的xtype值（全部小写）。对于代理类的别名设置也基本类似，但前缀为proxy，后面加上代理名称。

下一步，配置代理类型（#2）。staticData包里所有的存储器都通过Ajax调用方式与服务器端通信。

之后，配置reader属性项，该属性描述从服务器端读取数据的方式：

```
reader: {
    type: 'json',
    messageProperty: 'msg',
    root: 'data'
}
```

同以往一样，我们使用JSON作为服务器端与Ext JS间交换数据的格式。我们通过JSON对象的data属性包装模型集合，还打算通过msg属性（从服务器端）发送额外信息给Ext JS。

我们讲解了怎样通过reader从服务器端读取数据，接下来看看如何通过writer返回数据给服务器端：

```
writer: {
    type: 'json',          // #3
    writeAllFields: true, // #4
    encode: true,          // #5
    allowSingle: false,    // #6
    root: 'data'           // #7
}
```

我们依旧通过JSON对象返回数据给服务器端（#3）。以下是返回给服务器端的数据格式样例（`Actor`模型）：

```
{
    "data": [{
        "last_update": "2013-01-28 13:42:00",
        "actor_id": "1",
        "first_name": "PENELOPE",
        "last_name": "GUINESS"
    }]
}
```

`writeAllFields`配置项（值为`true`）表示Ext JS总是返回所有模型字段给服务器端，不管字段是否有更新（#4）。我们以`Actor`模型为例，假设我们编辑了`Actor`模型类的`last_name`列，就应该将`Actor`模型信息返回给服务器端执行`UPDATE`数据库操作。由于`writeAllFields`设为`true`，就会返回`id`、`first_name`（尚未更新）以及`last_name`（有了新值）等字段。对于所有字段，执行以下`UPDATE`操作（即使所有的列都没有更新过）：

```
UPDATE Actor
SET
  first_name = {first_name: },
  last_name = {last_name: },
  last_update = {last_update: CURRENT_TIMESTAMP}
WHERE actor_id= {actor_id};
```

如果将`writeAllFields`设置为`false`，我们就得核实返回服务器端的字段，并动态地创建`SET`语句。因此，为了节省时间，我们将`writeAllFields`设置为`true`。

之后，设置encode属性配置项为`true`（#5）。这意味着我们将发送经过编码的JSON格式数据，而非`data`原样格式。

接着将`allowSingle`属性配置项设置为`false`（#6）。也就是我们总是把发送给服务器端的信息用方括号“[]”包装起来，即使是只发送一个模型实例信息。如果我们未考虑到服务器端的某种情形（后面会提到），就会碰到令人头疼的事情。

当我们发送一个模型实例到服务器端（执行创建、修改或删除操作）时，并不能保证顺利完成不出错（我们希望如此，但往往事不遂人愿，错误、异常很可能发生）。当通过存储器编辑或

创建一条记录时，模型实例被标识为“脏”。这样当执行sync方法时存储器就知道需要发送哪些数据到服务器端。如果动作顺利完成，返回确认信息（acknowledgment，ACK）到Ext JS端说明动作完成，同时存储器从模型实例中移除“脏”标识。如果动作不成功，服务器端将返回一个异常，或者success值置为false，存储器不会移除要保存的模型实例的“脏”标识。

然而，假设我们再编辑另一个模型，接着保存它。如果前面的模型未保存，当我们再次执行存储器的sync方法时，存储器就会发送两个模型实例至服务器端，并且将信息包装在方括号内。如果不把信息包装在方括号内，就发送多个模型实例至服务器端，这种情形下，服务器端就会抛出错误或异常。因此为了避开这种验证模式，我们用方括号包装数据；这种方式下，我们只需编写一段代码即可。

最后，把发送信息包装在data属性里（#7），跟我们从服务器端读取数据一样。

接下来，我们将在代理中添加安全性配置项。改变读取动作的动作方法：

```
actionMethods: {
    create: "POST",
    read: "POST", //改为POST
    update: "POST",
    destroy: "POST"
}
```

当使用Ajax方式与服务器端交换数据时，创建、修改和销毁（删除）等动作通过POST方法发送信息至服务器端。读取动作默认使用GET方法。但是，我们将更改读取动作为POST方法。这样，所有的参数都通过POST方法传送，而非查询字符串方式（将参数添加在URL里，比如 http://localhost/masteringextjs/php/staticData/actor/create.php?entity=Actor）。

在继续之前，思考一下为什么要创建代理超类。我们有5个模型类：Actor、Category、Language、City和Country。我们希望这样执行SELECT查询：

```
SELECT * FROM Actor
```

我们要从名为Actor的表中获取所有记录。这里作为静态数据模型，并不使用远程排序、过滤或分页，因此，使用简单的SELECT语句就可以。我们也可以把Actor替换为其他表名：Category、Language、City或Country。如果我们能够发送表名至获取记录的服务器端就更好了，这样就可以用同一URL获取数据库表记录，这正是我们要做的。

对删除、修改和创建记录等操作，当用简单PHP代码替代面向对象PHP代码方式时，想要通过同一代码执行DELETE、UPDATE和INSERT语句是不可能的。但我们可以使用同一URL，获取表名，重定向至特定代码。

也就是说，我们可以使用同一URL处理所有的CRUD操作：创建、读取、修改和销毁（删除）：

```
api: {
    read    : 'php/staticData/list.php',
    create  : 'php/staticData/create.php',
    update  : 'php/staticData/update.php',
    destroy : 'php/staticData/delete.php'
}
```

最后，添加一个异常监听器，在异常发生时显示信息给用户。

```
listeners: {
    exception: function(proxy, response, operation){
        Ext.MessageBox.show({
            title: 'REMOTE EXCEPTION',
            msg: operation.getError(),
            icon: Ext.MessageBox.ERROR,
            buttons: Ext.Msg.OK
        });
    }
}
```

现在，我们所有的配置都完成了！这样做最大的好处就是：集中编写一段代码就能实现所有特定存储器的此功能，而不需要在每个特定存储器中重复编写这些代码。

6.3.3 特定存储器

下面就来实现Actors、Categories、Languages、Cities和Countries存储器。

先创建Actors存储器：

```
Ext.define('Packt.store.staticData.Actors', {
    extend: 'Packt.store.staticData.Abstract', // #1

    requires: [
        'Packt.model.staticData.Actor',   // #2
        'Packt.proxy.StaticData'          // #3
    ],
    model: 'Packt.model.staticData.Actor', // #4

    proxy: {
        type: 'staticdataproxy',         // #5
        extraParams: {
            entity: 'Actor'              // #6
        }
    }
});
```

定义存储器后，我们需指定扩展存储器类。由于我们使用了存储器超类，可以直接扩展自这个超类（#1，超类扩展自Ext.data.Store类）。

接下来，我们声明requires属性配置项（#2）。通常情况下需要声明模型类（#4）。由于我们还使用了静态数据代理类的xtype（#5），就需要事先在内存中加载此类（#3），让Ext JS能够顺利通过其xtype实例化该代理类（Ext JS实例化代理类时，如果这个代理类不是原生代理，也未加载至内存，Ext JS将因无法识别这个类而抛出异常）。

最后，需要有一个proxy声明。代理的type是前面创建的静态数据代理。针对每个特定存储器的唯一属性配置项是entity（#6），其值为取回记录对应的数据库表名称。这个参数将在每一次从存储器至服务器端的请求中发送。

要着重关注创建、删除、读取和修改读写器的URL，以及其他并未在此列出但常在代理类中配置的信息。因为我们已不用再处理，所有这些都在代理超类里实现了。

6.4　创建菜单项

在创建界面之前，我们需要在动态菜单中添加菜单项。针对Menu表执行以下脚本代码：

```
INSERT INTO `sakila`.`Menu` (`id`, `text`, `iconCls`)
VALUES ('4', 'staticData', 'menu_staticdata');
```

首先，添加一个名为Static Data的菜单模块。

```
INSERT INTO `sakila`.`Menu` (`text`, `iconCls`, `parent_id`, `className`)
VALUES
('actor', 'menu_actor', '4', 'actorsgrid'),
('category', 'menu_category', '4', 'categoriesgrid'),
('language', 'menu_language', '4', 'languagesgrid'),
('city', 'menu_city', '4', 'citiesgrid'),
('country', 'menu_country', '4', 'countriesgrid');
```

接着，添加Static Data菜单的每个子菜单项。

Menu表如下图所示：

Menu

Filter:　Edit:　Export:

id	text	iconCls	parent_id	className
1	menu1	menu_admin	NULL	NULL
2	menu11	menu_groups	1	panel
3	menu12	menu_users	1	users
4	staticData	menu_staticdata	NULL	NULL
5	actors	menu_actor	4	actorsgrid
6	categories	menu_category	4	categoriesgrid
7	languages	menu_language	4	languagesgrid
8	cities	menu_city	4	citiesgrid
9	countries	menu_country	4	countriesgrid

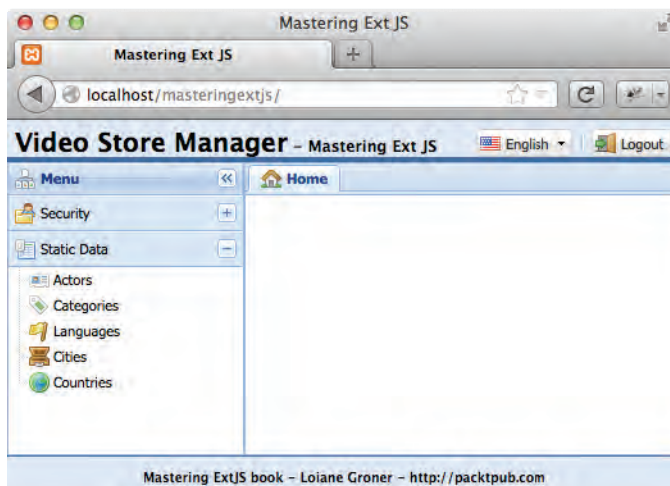
因为我们登录时输入的用户名是admin（管理员）用户组的一员，我们也手动向数据库中添加该用户组的所有权限：

```
INSERT INTO `sakila`.`Permissions` (`Menu_id`, `Group_id`)
VALUES
('4', '1'),
('5', '1'),
('6', '1'),
('7', '1'),
('8', '1'),
('9', '1');
```

在语言转换文件（en.js、es.js和pt.js）里，我们将添加以下属性及其转换信息：

```
staticData: 'Static Data',
actors: 'Actors',
categories: 'Categories',
languages: 'Languages',
cities: 'Cities',
countries: 'Countries'
```

输出效果如下图所示：



6.5　创建重用的抽象网格面板

现在该来实现视图了。我们将实现5个视图，分别针对Actors（演员）、Categories（类别）、Languages（用户语言）、Cities（所在城市）以及Countries（国家）等各表进行CRUD操作。

Actors表管理界面实现后的最终效果图如下：

Home | Actors | Categories

Add | Save Changes | Cancel Changes | Clear Filters

Search　< >　Regular expression　Case sensitive

Actor Id	First Name	Last Name	Last Update
1	PENELOPE	GUINESS	2006-02-15 04:34:33
2	NICK	WAHLBERG	2006-02-15 04:34:33
3	ED	CHASE	2006-02-15 04:34:33
4	JENNIFER	DAVIS	2006-02-15 04:34:33
5	JOHNNY	LOLLOBRIGIDA	2006-02-15 04:34:33
6	BETTE	NICHOLSON	2006-02-15 04:34:33
7	GRACE	MOSTEL	2006-02-15 04:34:33
8	MATTHEW	JOHANSSON	2006-02-15 04:34:33

Nothing Found

Categories表管理界面实现后的最终效果图如下：

Home | Actors | Categories

Add | Save Changes | Cancel Changes | Clear Filters

Search　< >　Regular expression　Case sensitive

Category Id	Category Name	Last Update
1	Action	2006-02-15 04:46:27
2	Animation	2006-02-15 04:46:27
3	Children	2006-02-15 04:46:27
4	Classics	2006-02-15 04:46:27
5	Comedy	2006-02-15 04:46:27
6	Documentary	2006-02-15 04:46:27
7	Drama	2006-02-15 04:46:27
8	Family	2006-02-15 04:46:27

Nothing Found

发现两个界面间的相似之处了吗？我们再次观察一下：

Home | Actors

Add | Save Changes | Cancel Changes | Clear Filters　1

Search　< >　Regular expression　Case sensitive　2

Actor Id | First Name | Last Name | Last Update

71 ADAM　　2006-02-15 04:34:33

132 ADAM　　2006-02-15 04:34:33

142 JADA　　2006-02-15 04:34:33

Sort Ascending

Sort Descending　5

Columns　4　3

Filters　ad

Nothing Found

最顶层的工具栏都一样（1）；都有一个即席搜索功能（Search，2）、过滤插件（Filters，4）、

最后修改列（Last Update）以及操作列（action，3）。再进一步，两个网格面板都可以通过单元格编辑插件进行编辑。两个界面的不同点是在于它们的列（5）。这是不是意味着我们可以创建具备通用功能的网格面板超类，并通过继承达到重用代码的目的呢？是的！

下面我们就创建网格面板超类。首先，创建新的Packt.view.staticData.AbstractGrid类：

```
Ext.define('Packt.view.staticData.AbstractGrid', {
    extend: 'Ext.ux.LiveSearchGridPanel', // #1
    alias: 'widget.staticdatagrid',

    columnLines: true, // #2
    viewConfig: {
        stripeRows: true // #3
    },
});
```

该类扩展自Ext.ux.LiveSearchGridPanel类，而非Ext.grid.Panel类。Ext.ux.LiveSearchGridPanel类本身扩展自extends the Ext.grid.Panel类，并添加了即席搜索工具栏（2）。LiveSearchGridPanel类是随Ext JS SDK一同发布的插件。你可以在sdk/examples/ux目录下找到它。我们把ux复制到extjs目录下（extjs/ux）。

其中，#2和#3配置项呈现单元格边线，并用白色和浅灰色两种颜色交替作为行背景色。

6

第一次使用Ext.ux命名空间时，需要告诉加载器类在哪儿能找到这些文件。打开app.js文件，在加载器类里添加映射：

```
Ext.Loader.setConfig({
    enabled: true,
    paths: {
        Ext: '.',
        'Ext.ux': 'extjs/ux',
        'Packt.util': 'app/util'
    }
});
```

接下来是创建一个initComponent方法。我们创建超类时，一些配置项将被覆写，而另一些不用。通常，要覆写的内容在类中声明为配置项。不用覆写的内容，在initComponent方法里声明。由于有些配置项不用覆写，我们就在initComponent方法里声明它们：

```
initComponent: function() {
    var me = this;

    me.selModel = {
        selType: 'cellmodel' // #4
    };

    me.plugins = [
        Ext.create('Ext.grid.plugin.CellEditing', { // #5
            clicksToEdit: 1,
```

```
            pluginId: 'cellplugin'
        })
    ];

    me.features = [
        Ext.create('Ext.ux.grid.FiltersFeature', { // #6
            local: true
        })
    ];

    //停驻项

    //列

    me.callParent(arguments);
}
```



me和this的比较

如果你看过一些互联网上的Ext JS代码，就会发现很多开发者用me替代this。为什么这样做呢？一旦this被大量使用，你就会考虑用me了，因为这将节省16比特并能更好地压缩（在构建产品后，代码也将被压缩）。同时，me也可用于我们需要在另一个函数里管理当前上下文作用域的情形。

我们还可以定义用户选择网格面板信息的模式：默认配置是行选择模式。由于我们希望用户能够编辑单元格，因此采用单元格选择模式（#4），并使用单元格编辑插件（#5，已在Ext JS SDK里了）。

为了过滤信息（即席搜索只高亮匹配记录），我们将使用过滤特性（#6）。过滤特性不在Ext JS SDK里，你可以在ux目录下找到它。

接着声明停驻项。因为所有网格面板都有同样的工具栏，我们可以在超类中声明它（initComponent方法里）：

```
me.dockedItems = [
    {
        xtype: 'toolbar',
        dock: 'top',
        itemId: 'topToolbar',
        items: [
            {
                xtype: 'button',
                itemId: 'add',
                text: 'Add',
                iconCls: 'add'
            },
            {
```

```
                xtype: 'tbseparator'
            },
            {
                xtype: 'button',
                itemId: 'save',
                text: 'Save Changes',
                iconCls: 'save_all'
            },
            {
                xtype: 'button',
                itemId: 'cancel',
                text: 'Cancel Changes',
                iconCls: 'cancel'
            },
            {
                xtype: 'tbseparator'
            },
            {
                xtype: 'button',
                itemId: 'clearFilter',
                text: 'Clear Filters',
                iconCls: 'clear_filter'
            }
        ]
    }
];
```

这样，我们就有了Add（添加）、Save Changes（保存修改）、Cancel Changes（取消修改）以及Clear Filter（清除过滤）等按钮。

最后，我们添加所有界面都有的两列（`Last Update`列，以及操作列`Delete`），这两列后续会跟特定网格面板的其他列一同呈现：

```
me.columns = Ext.Array.merge(me.columns,
    [{
        xtype    : 'datecolumn',
        text     : 'Last Update',
        width    : 120,
        dataIndex: 'last_update',
        format: 'Y-m-j H:i:s',
        filter: true
    },
    {
        xtype: 'actioncolumn',
        width: 30,
        sortable: false,
        menuDisabled: true,
        items: [
            {
                handler: function(view, rowIndex, colIndex, item, e) {
                    this.fireEvent('itemclick', this, 'delete', view, rowIndex,
colIndex, item, e);
```

6

```
                    },
                    iconCls: 'delete',
                    tooltip: 'Delete'
                }
            ]
        }]
    );
```

6.5.1　用MVC架构模式处理操作列

我们再看一下网格面板超类里的操作列声明部分，特别是handle属性配置项：

```
handler: function(view, rowIndex, colIndex, item, e) {
    this.fireEvent('itemclick', this, 'delete', view, rowIndex, colIndex, item, e);
}
```

在操作列添加的“按钮”并不是组件(因为它不是真正的按钮，只是图片或图标)，因此，就无法在控制器里监听它们的事件。但是，如果在handle属性配置项进行事件处理也就意味着无法遵循MVC架构。因此，我们需要在操作列（它是个组件）上触发一个自定义事件并传递表示点击动作名称（delete）的额外参数。这样，我们就可以在控制器里监听这个事件了。

6.5.2　在操作列用**iconCls**属性取代**icon**属性

通常，我们声明一个操作列子项时，会设置icon属性配置项为操作列所用图标的路径，但这种方式并不好。试想若是别的地方也用到这个图标而我们打算做些改动，那就得对所有使用此图标的地方分别进行手工设置。

设置按钮图标时，我们可以在CSS文件中创建一个样式，并设置iconCls属性配置项。这种方式为日后的改动提供了便利，只需改动样式，所有用到该图标的地方都会产生相应调整。

能够在操作列使用同样的方式吗？能用iconCls属性设置取代icon属性（路径设置）吗？没问题！接下来看看怎样实现它。

首先，我们需要创建一个图标样式。我们使用前面的删除按钮iconCls设置：

```
.delete {
  background-image:url('../icons/delete.png') !important;
}
```

接下来，在CSS文件里添加一个新样式：

```
.x-action-col-cell img {
    height: 16px;
    width: 16px;
    cursor: pointer;
}
```

以上就是样式设置的全部！接下来，在操作列子项上应用iconCls配置项：

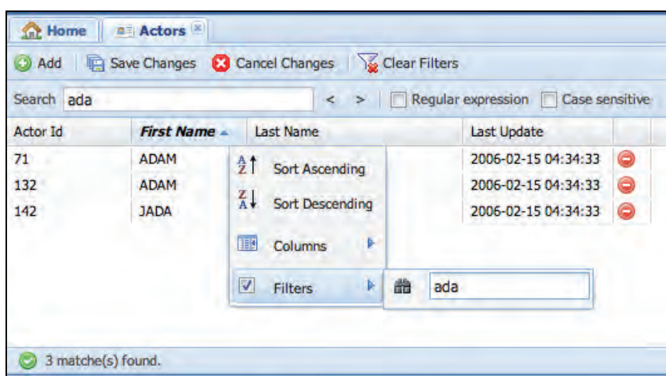
```
{
    xtype: 'actioncolumn',
    width: 30,
    sortable: false,
    menuDisabled: true,
    items: [
        {
            iconCls: 'delete',
            tooltip: 'Delete'
        }
    ]
}
```

6.5.3　比较即席搜索插件与过滤插件

即席搜索插件（Live Search）与过滤插件（Filter）都能帮助用户快速找到信息。本项目中两者都会用到。

即席搜索插件搜索网格面板所有列的匹配结果。这种插件执行本地搜索，也就是说，如果使用分页工具栏，插件就无法正常工作。本例中，将实时显示所有数据库记录，因此插件可以正常工作。比如，搜索“ada”会得到以下输出：

过滤插件在存储器上执行过滤，因此将匹配结果显示给用户：



6.5.4　对应每张数据库表的特定网格面板

实现控制器功能前的最后一步是实现特定网格面板。我们在前面已经完成了覆盖大部分功能的网格面板超类。现在我们需要为每个特定网格面板声明个性化配置项。

我们需要创建5个扩展自`Packt.view.staticData.AbstractGrid`类的特定网格面板：

- `Packt.view.staticData.Actors`
- `Packt.view.staticData.Categories`
- `Packt.view.staticData.Languages`
- `Packt.view.staticData.Cities`
- `Packt.view.staticData.Countries`

第一步，创建Actors网格面板：

```
Ext.define('Packt.view.staticData.Actors', {
    extend: 'Packt.view.staticData.AbstractGrid',
    alias: 'widget.actorsgrid',

    store: 'staticData.Actors', // #1

    columns: [
        {
            text: 'Actor Id',
            width: 100,
            dataIndex: 'actor_id',
            filter: {
                type: 'numeric' // #2
            }
        },
        {
            text: 'First Name',
            flex: 1,
            dataIndex: 'first_name',
```

```
                editor: {
                    allowBlank: false, // #3
                    maxLength: 45      // #4
                },
                filter: {
                    type: 'string'     // #5
                }
            },
            {
                text: 'Last Name',
                width: 200,
                dataIndex: 'last_name',
                editor: {
                    allowBlank: false, // #6
                    maxLength: 45      // #7
                },
                filter: {
                    type: 'string'     // #8
                }
            }
        ]
});
```

首先声明存储器（#1），使用`Actors`存储器。由于`Actors`存储器在staticData目录下（store/staticData），所以需要带上子目录名称，否则，Ext JS会认为存储器在app/store目录里，如此就会出错。

6

接下来声明Actors网格面板的特定列（由于`Last Update`列以及`Delete`操作列已在网格面板超类中声明过了，这里不再重复声明）。

要注意每列的`editor`和`filter`属性配置项。`editor`属性配置作用于编辑操作，我们只在用户需要编辑的列上进行属性设置；在需要执行过滤的列上设置`filter`属性。

例如，`id`列并不需要编辑，它是MySQL数据库表里的自增序列字段，因此，不需要设置`editor`属性。但是，用户可以基于ID进行信息过滤，因此，我们设置了`filter`属性（#2）。

对另外两列：`first_name`和`last_name`，用户可以进行编辑，因此添加了`editor`属性。还可以像处理表单字段那样，设置客户端验证，比如：两个字段都不能为空（#3和#6）、可输入的最大字符数（#4和#7）。

最后，所有的列都渲染为文本类型（`string`），并设置`first_name`和`last_name`两个列的`filter`属性（#5和#8）。其他的所有功能都将在网格面板超类中实现。

6.6 通用控制器

现在我们来实现静态数据模块的最后一部分。我们的目标是在一个控制器类里尽可能实现所

有界面的通用功能，但不包括个性化功能的实现。

先从控制器的基础开始。创建一个新类Packt.controller.staticData.AbstractController：

```
Ext.define('Packt.controller.staticData.AbstractController', {
    extend: 'Ext.app.Controller',

    requires: [  // #1
        'Packt.util.Util'
    ],

    stores: [    // #2
        'staticData.Actors',
        'staticData.Categories',
        'staticData.Cities',
        'staticData.Countries',
        'staticData.Languages'
    ],

    views: [    // #3
        'staticData.AbstractGrid',
        'staticData.Actors',
        'staticData.Categories',
        'staticData.Cities',
        'staticData.Countries',
        'staticData.Languages'
    ],

 init: function(application) {
     this.control({
     });
 }
});
```

现在，声明requires（#1 ，在此通过一些方法使用Util类）、stores（#2 ，在此列出模块的所有存储器）以及views（#3 ，在此列出模块的所有视图，包括抽象网格面板）。

通常，我们还要实现init函数和this.control属性以在其中监听所有的组件事件。

6.6.1　在网格渲染时加载网格面板

我们希望在需要时加载存储器。当用户点击动态菜单，应用程序打开界面时，网格面板被渲染。当网格面板被渲染时，我们希望加载存储器。而达成此目标需要监听网格面板的render（渲染）事件。

现在来看看最出彩的部分吧。当声明控制器所监听组件的选择器时，不能使用grid或gridpanel作为xrtpe，因为我们只关心静态数据模块的网格面板。我们也不希望使用每个静态

数据网格面板的xtype（actorsgrid、categoriesgrid、anguagesgrid、citiesgrid和countriesgrid），因为我们要实现的是所有组件通用的功能代码。好消息是前面我们通过继承方式实现了网格面板超类（Packt.view.staticData.AbstractGrid），其xtype为staticdatagrid。

比如，我们监听一个Actors网格面板的事件，可以使用属性值为actorsgrid的xtype、超类的xtype（staticdatagrid）或属性值为grid/'gridpanel'的xtype。继承就是这么美妙！

因此，我们回到选择器，使用staticdatagrid并监听render事件：

```
"staticdatagrid": {
    render: this.render
}
```

接下来声明render方法：

```
render: function(component, options) {
    component.getStore().load();
}
```

现在有几种方式可供选择。我们可以在正在渲染的特定组件中获取对应存储器并加载它。但在这里，由于网格面板是以一个参数的方式传进render事件（component），我们就可以通过getStore方法获取特定网格面板对应的存储器，然后调用load方法。

这种方式下，加载Actors网格面板时，相应的存储器（Actors存储器）也将被加载；加载Categories网格面板时，相应的Categories存储器也将被加载，以此类推。

我们只用了一行代码，就实现了通用的逻辑代码，为所有静态数据网格面板提供了通用功能。

6.6.2 在网格面板上添加记录

每个静态数据网格面板的工具栏上都有个添加（Add）按钮，点击这个按钮，可以添加一个模型到存储器里（相应的，网格面板也会添加一条记录），并进入编辑状态让用户填充信息后进行保存[点击Save Changes（保存修改）按钮]。

首先，监听添加按钮的click事件：

```
"staticdatagrid button#add": {
    click: this.onButtonClickAdd
}
```

然后，实现onButtonClickAdd方法：

```
onButtonClickAdd: function (button, e, options) {
    var grid= button.up('staticdatagrid'), // #1
    store = grid.getStore(),                // #2
```

```
        modelName = store.getProxy().getModel().modelName, // #3
        cellEditing = grid.getPlugin('cellplugin'); // #4

        store.insert(0, Ext.create(modelName, {  // #5
            last_update: new Date()              // #6
        }));

        cellEditing.startEditByPosition({row: 0, column: 1}); // #7
    }
```

通过参数，我们只能得到按钮引用。事实上，我们需要获取网格面板的引用，因此，通过up方法得到它（#1）。再一次使用网格面板超类的xtype作为选择器（staticdatagrid），这样就能够使代码更加通用。

当我们获得网格面板引用后，就可以通过getStore方法获取存储器引用（#2）。

我们通过模型名称实例化模型（#5），以便在存储器的开始位置插入记录（在网格面板的第一行）。继续从通用代码角度考虑，可以通过存储器的proxy获取模型名称（#3）。当实例化模型时，可以传入一些配置项。我们希望Last Update列也被更新，那么就可以将其作为配置项传入并赋以最新时间（#6）。

最后，需要聚焦该行的某个单元格使之成为活动单元格，这样用户就知道这个单元格可以编辑。因此，我们聚焦第一行第二列（第一列是id列，不可编辑）的单元格（#7）。在这之前，需要获取单元格编辑插件的引用，我们可以通过传入pluginId参数给getPlugin方法的方式实现（#4）。

还记得吗？我们在Packt.view.staticData.AbstractGrid类的CellEditing单元格编辑插件中声明了pluginId属性：

```
Ext.create('Ext.grid.plugin.CellEditing', {
    clicksToEdit: 1,
    pluginId: 'cellplugin'
})
```

6.6.3　编辑存在记录

CellEditing插件会自动保存单元格的编辑内容。但是，当用户点击单元格进入编辑状态到完成编辑，我们需要将Last Update列的值更新为最新时间。

CellEditing插件有个edit事件能够满足我们所需。不幸的是，控制器无法监听插件事件；幸运的是，网格面板类也会触发这个事件（CellEditing插件传递这个事件给网格面板），因此可以在网格面板类监听该事件。我们是在网格面板超类声明事件的，因此在其中添加edit事件，如以下粗体所示：

```
"staticdatagrid": {
    render: this.render,
    edit: this.onEdit
}
```

接下来，我们实现onEdit方法：

```
onEdit: function(editor, context, options) {
    context.record.set('last_update', new Date());
}
```

第二个参数是事件（context）。通过它，可以获取用户编辑的模型实例（record），之后设置last_update字段为当前时间。

6.6.4 删除：在控制器中处理操作列

现在，读取、创建和修改操作都已实现，下面实现删除操作。我们用操作列的删除项来取代删除按钮。在前面的章节中，我么了解了如何触发操作列子项的事件以便在控制器里处理该事件。我们无法监听操作列子项触发的事件，但可以监听操作列触发的事件：

```
"staticdatagrid actioncolumn": {
    itemclick: this.handleActionColumn
}
```


现在，我们来实现handleActionColumn方法：

```
handleActionColumn: function(column, action, view, rowIndex, colIndex, item, e) {
        var store = view.up('staticdatagrid').getStore(),
        rec = store.getAt(rowIndex);

        if (action == 'delete'){
            store.remove(rec);
            Ext.Msg.alert('Delete', 'Save the changes to persist the removed
record.');
        }
    }
```

这是一个自定义事件，需要获取操作列子项传入的参数。

首先，获取存储器以及用户点击删除的记录。之后，通过第二个参数（操作列子项动作名称）获悉哪些子项（动作）触发了事件。如果动作是delete，就从存储器移除这条记录，并提示用户点击Save Changes按钮提交变更，这样存储器中的模型与服务器端数据将保持同步。

6.6.5 保存变更

用户执行修改、删除或创建操作之后，变更过信息的单元格都有个标识（“脏”标识告知存储器修改了哪个模型），如下图所示：

Home　Actors

Add　Save Changes　Cancel Changes　Clear Filters

Search　<　>　Regular expression　Case sensitive

Actor Id	First Name	Last Name	Last Update
	First Name	Last Name	2013-02-3 11:05:00
1	PENELOPE	GUINESS	2006-02-15 04:34:33
2	NICK-updated	WAHLBERG	2013-02-3 11:04:05
3	ED	CHASE	2006-02-15 04:34:33
4	JENNIFER	DAVIS-updated	2013-02-3 11:05:00
5	JOHNNY	LOLLOBRIGIDA	2006-02-15 04:34:33
6	BETTE	NICHOLSON	2006-02-15 04:34:33
7	GRACE	MOSTEL	2006-02-15 04:34:33

Nothing Found

采用同样方式，我们可以通过保存变更信息（提交）在MySQL数据库表中执行修改。这就是创建Save Changes按钮的原因，这样就可以立刻将变更信息同步到服务器端。

因此，首先在this.control里添加一个监听器：

```
"staticdatagrid button#save": {
    click: this.onButtonClickSave
}
```

接下来实现onButtonClickSave方法：

```
onButtonClickSave: function (button, e, options) {
  button.up('staticdatagrid').getStore().sync();
}
```

这个方法的代码简洁明了，我们只需获取网格面板的存储器，然后调用sync方法即可。

autoSync属性配置项

存储器有个autoSync属性配置项。默认值为false，如果设为true，存储器会在检测到变更后自动与服务器端同步数据。这一点说不上好坏，取决于我们怎么利用它。

如何设置autoSync属性取决于我们的变更频率。例如，对Categories（类别）界面，用户不会频繁对类别进行增删改，变更就很少，这样将autoSync设置为true就没问题。

现在假设用户需要频繁增删改记录。这时，如果将autoSync设置为true就非常危险。按这种频率，Ext JS会发送大量查询到服务器端，服务器端就有可能将其当作DoS攻击（Denial-of-Service attack，拒绝服务攻击）而中断这些请求。这是因为对于每个增删改操作，Ext JS都会发送一个请求给服务器端，这就是不同于将autoSync设置为false时的地方。如果有大量数据要执行增删改操作，这种方式的弊端可想而知，因此，当设置autoSync为true时就要非常小心。



若希望了解更多关于DoS攻击的内容，请参考http://en.wikipedia.org/wiki/Denial-of-service_attack。

6.6.6　取消变更

就像用户可以保存变更（提交）一样，用户也能取消变更（回滚）。我们要做的就是重新加载存储器获取服务器端最新信息，这样客户端用户进行的所有变更都将失效。

因此，我们需要监听该事件：

```
"staticdatagrid button#cancel": {
    click: this.onButtonClickCancel
}
```

并实现相应方法：

```
onButtonClickCancel: function (button, e, options) {
  button.up('staticdatagrid').getStore().reload();
}
```

如果你愿意，还可以添加一条用户是否确定回滚变更的提示。通过存储器调用`reload`方法可以完成回滚操作。

6.6.7　清除过滤器

在网格面板上使用Filters插件时，一切工作都按照我们的意愿进行得很顺利（执行本地搜索的情况下）。但还有一件事情我们没做：提供随时清除过滤器的选项。因此需要实现一个Clear Filter（清除过滤器）按钮。

首先监听有关事件：

```
"staticdatagrid button#clearFilter": {
    click: this.onButtonClickClearFilter
}
```

然后实现该方法：

```
onButtonClickClearFilter: function (button, e, options) {
  button.up('staticdatagrid').filters.clearFilters();
}
```

当使用过滤插件时，通过网格面板获取`filters`属性。然后，我们只需调用`clearFilters`方法即可。这样就会清除被过滤各列的过滤值，并清除存储器中的过滤器。

6.6.8　在控制器中监听存储器事件

最后，监听存储器的write事件。我们已经为代理添加了exception异常监听器，现在来添加成功情形的监听器。

第一步，在控制器中监听存储器事件。请注意，这里无法使用Ext JS 4.0.*x*和4.1.*x*版本，这个特性需要Ext JS 4.2.*x*+版本。

在控制器init函数中添加以下代码：

```
this.listen({
    store: {
        '#staticDataAbstract': {
            write: this.onStoreSync
        }
    }
});
```

我们可以在store属性里监听存储器事件。创建存储器超类时，我们说过需要在其中监听事件，所有子类都包括在内。这也是创建存储器超类的原因，这样就不需要监听每个特定静态数据存储器类的事件了。

存储器任何时候收到服务器端响应都会触发write事件。接下来实现该方法：

```
onStoreSync: function(store, operation, options){
  Packt.util.Alert.msg('Success!', 'Your changes have been saved.');
}
```

我们简单地显示一条信息，表明变更被保存了。请注意这个信息也是通用的，可被各个静态数据模块所用。

6.7　小结

在本章，我们介绍了如何实现看起来与MySQL数据库表编辑器很相似的界面。本章最重要的内容就是通过OOP的继承概念实现抽象类。通常，我们习惯于在服务器端语言中运用这个概念，比如PHP、Java、.NET等。而本章阐述了在Ext JS上运用这些概念的重要性。通过这种方式，我们可以重用大量代码，实现供多个组件使用的通用功能。

我们创建了抽象模型、存储器、视图以及控制器，并学会了创建自定义代理类；同时应用了网格面板单元格编辑插件、即席搜索插件以及网格面板过滤插件；还学习了通过存储器功能执行CRUD操作的方法，知道使用autoSync属性配置项时不加以小心就可能造成危险后果。另外，我们还学习了在控制器里创建自定义事件及处理操作列子项事件的方法。

接下来，我们将学习怎样实现内容管理模块（本章只是管理单个表）。我们将管理来自数据库表的信息（与应用业务逻辑相关联），以及它们在数据库中的各种关系。

第 7 章

内容管理

7

上一章我们实现了模仿数据库表编辑器界面的静态数据模块，但它基本上还只能对单表进行CRUD操作，并带点额外功能。本章将进一步了解管理表信息的复杂性。在实际应用程序中，我们管理的表信息总会和其他表发生关联，这些关联同样需要管理。这就是本章的内容：学习如何使用Ext JS创建界面管理复杂信息。

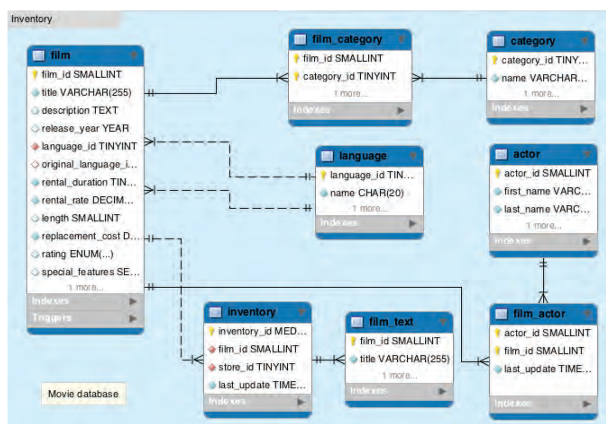
本章主要包括以下内容：

- 用Ext JS管理复杂信息；
- 处理多对多关系；
- 关联表单；
- 重用组件。

7.1　管理影片、客户和租借信息

Sakila数据库有4个主要模块：库存（Inventory）模块，包含影片及库存信息（每个商店可用于租借的影片数量）；客户数据（Customer Data）模块，包含客户信息；业务（Business）模块，包含商店、雇员、租借以及支付信息等（视库存模块和客户数据模块情况而定）；视图（Views）模块，包含图表展示用到的数据。

现在，我们先把精力放在库存模块、客户数据模块以及业务模块上，这些模块包含了应用的核心业务信息。先来看看库存模块，这个模块比其他两个模块拥有更多的数据库表：



根据Sakila数据库文档描述：

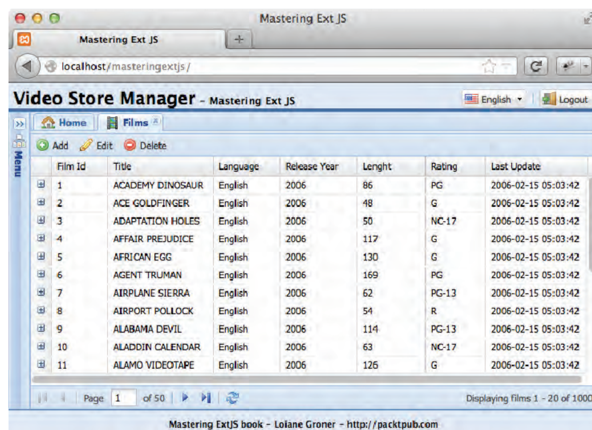
　　film（影片）表是商店里现有的所有影片的列表。每部影片实际存货数量反应在inventory（库存）表里。

　　film表参照language（语言）表，同时，被film_category表（影片类别映射表）、film_actor表（影片演员映射表）以及inventory表参照。

film表与category（类别）表、actor（演员）表之间都有一个多对多的关系。同时，film表跟language表之间有两个多对一关系。在前一章，我们已经实现了类别表、演员表和语言表的管理功能。现在，我们来实现film表与其他表间关联关系的管理功能。

先来简单看一下本章要实现的界面。虽然我们的Sakila数据库里有客户数据模块和业务模块，但本章我们只讨论与影片相关的内容。不用紧张，客户数据模块和业务模块遵循同样的方式，你可以从本书样例源代码中找到这两个模块的相关代码。

首先，需要一个界面列出我们拥有的影片：

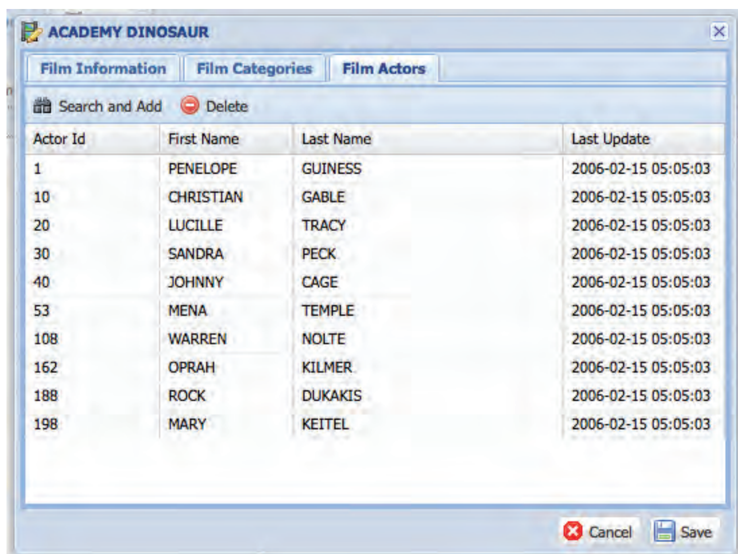


我们有可能希望在里面创建或编辑影片信息，因此，在窗体里创建表单面板供信息编辑使用：

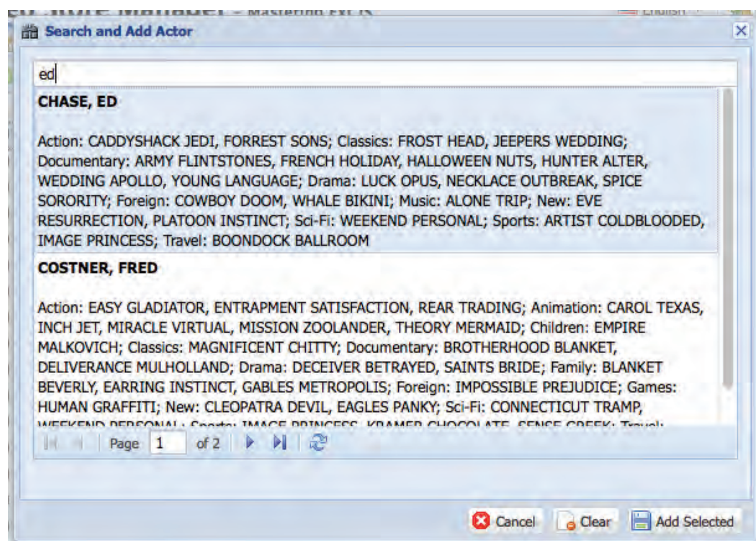
由于film表与category表是多对多关系，因此，我们需要在表单面板提供单独的标签页处理类别相关内容：

我们还打算添加更多的影片类别，因此，提供了Search and Add（搜索和添加）功能：

同样的，film表与actor表也是多对多关系，因此也要在表单面板上做类似处理：



如果我们想为影片添加更多的演员名单，可以使用Search and Add（搜索和添加演员名单）功能：



注意，我们对每个界面都进行了不同的方法处理。这样我们就可以在处理这些场景的过程中学习到更多的Ext JS使用方法。

好了，现在我们对本章的实现思路有了大体了解，接下来享受实践的乐趣吧！

7.2 呈现影片数据网格

首先，我们从基本实现入手。无论何时我们想实现多么复杂的界面，都应该从最简单的组件着手。我们要进行不断的修改并添加更多复杂功能，直至其能完全运行。因此，第一步我们需要创建模型和存储器来表示film表。这部分功能完成后，就实现了film表与category表、language表以及actor表的关联关系。

7.2.1 影片模型

首先，我们要创建表示film表的模型。先不用考虑它具有哪些关联关系。

我们来创建一个新类Packt.model.film.Film：

```
Ext.define('Packt.model.film.Film', {
    extend: 'Packt.model.sakila.Sakila',

    idProperty: 'film_id',

    fields: [
        { name: 'film_id' },
        { name: 'title', type: 'string' },
        { name: 'description', type: 'string'},
        { name: 'release_year', type: 'int'},
        { name: 'language_id'},
        { name: 'original_language_id'},
        { name: 'rental_duration', type: 'int'},
        { name: 'rental_rate', type: 'float'},
        { name: 'length', type: 'int'},
        { name: 'replacement_cost', type: 'float'},
        { name: 'rating'},
        { name: 'special_features'}
    ]
});
```

由于所有的Sakila数据库表都有last_update列，因此，我们扩展自Packt.model.sakila.Sakila类以避免在每个创建的模型（代表某个Sakila数据库表）中都声明这个字段。

声明的字段跟file数据库表字段保持一致。

7.2.2 影片存储器

下一步是创建加载影片集合的存储器。我们来创建新的存储器Packt.store.film.Films（如果你想遵循Sencha惯例，需谨记存储器名称总是模型名称的复数形式）：

```
Ext.define('Packt.store.film.Films', {
    extend: 'Ext.data.Store',
```

```
    requires: [
        'Packt.model.film.Film',
        'Packt.proxy.Sakila'
    ],

    model: 'Packt.model.film.Film',
    pageSize: 20, // #1

    storeId: 'films',

    proxy: {
        type: 'sakila', // #2
        url: 'php/inventory/list.php'
    }
});
```

在这个存储器里，我们像往常一样声明了模型，同时还声明pageSize属性值为20（#1），这意味着我们将在影片数据网格中用到分页工具栏，并一次获取20部影片显示在网格面板中。

注意，我们将代理类型type设置为sakila，也就是说，并未采用原生的代理。为什么需要一个自定义代理呢？本书中，大多数情况下，我们都会声明一个代理，代理类型type通常为Ajax，且代理中读写器（reader和writer）用到的设置项通常是一样的。因此，为了避免在每次声明代理时重复声明同样的配置项，我们可以创建自己的自定义代理类。

记得要在app/proxy目录下创建这个代理类：

```
Ext.define('Packt.proxy.Sakila', {
    extend: 'Ext.data.proxy.Ajax',
    alias: 'proxy.sakila',

    type: 'ajax',

    reader: {
        type: 'json',
        messageProperty: 'msg',
        root: 'data'
    },

    writer: {
        type: 'json',
        writeAllFields: true,
        encode: true,
        allowSingle: false,
        root: 'data'
    },

    listeners: {
        exception: function(proxy, response, operation){
            Ext.MessageBox.show({
                title: 'REMOTE EXCEPTION',
```

```
                msg: operation.getError(),
                icon: Ext.MessageBox.ERROR,
                buttons: Ext.Msg.OK
            });
        }
    }
});
```

要在存储器中使用这个代理类，我们只需在requires属性声明里添加这个代理类，并使用这个代理类型。如果你愿意，可以按此思路，重构前面几章已创建的有关存储器。

7.2.3　带分页功能的影片数据网格

我们现在有了模型和存储器，接下来需要创建影片数据网格：

```
Ext.define('Packt.view.film.Films', {
    extend: 'Packt.view.sakila.SakilaGrid', // #1
    alias: 'widget.filmsgrid',

    requires: [
        'Ext.ux.RowExpander' // #2
    ],

    store: 'film.Films',

    columns: [
        {
            text: 'Film Id',
            width: 100,
            dataIndex: 'film_id'
        },
        {
            text: 'Title',
            flex: 1,
            dataIndex: 'title'
        },
        {
            text: 'Language',
            width: 100,
            dataIndex: 'language_id', // #3
            renderer: function(value, metaData, record ){
                var languagesStore = Ext.getStore('languages');
var lang = languagesStore.findRecord('language_id', value);
                return lang != null ? lang.get('name') : value;
            }
        },
        {
            text: 'Release Year',
            width: 90,
            dataIndex: 'release_year'
        },
```

```
        {
            text: 'Lenght',
            width: 80,
            dataIndex: 'length'
        },
        {
            text: 'Rating',
            width: 70,
            dataIndex: 'rating'
        }
    ],

    dockedItems: [
        {
            dock: 'bottom',
            xtype: 'pagingtoolbar', // #4
            store: 'film.Films',
            displayInfo: true,
            displayMsg: 'Displaying films {0} - {1} of {2}',
            emptyMsg: "No films to display"
        }
    ],

    plugins: [{ // #5
        ptype: 'rowexpander',
        rowBodyTpl : [
            '<p><b>Description:</b> {description}</p><br>',
            '<p><b>Special Features:</b> {special_features}</p><br>',
            '<p><b>Rental Duration:</b> {rental_duration}</p><br>',
            '<p><b>Rental Rate:</b> {rental_rate}</p><br>',
            '<p><b>Replacement Cost:</b> {replacement_cost}</p><br>'
        ]
    }]
});
```

由于应用程序慢慢变大了，我们发现在相同组件里大量使用了某些配置项。例如，大多数的网格面板都使用了带有添加、编辑和删除按钮的工具栏；所有的Sakila数据库表都有最后修改列，因此，这列对于用来列出Sakila数据库表信息的网格面板同样适用。出于这样的考虑，我们可以创建一个网格面板超类（正如针对静态数据模块所做的）。因此，对于影片网格面板，将扩展自SakilaGrid类（#1，后续会创建）。

接下来，我们声明requires为RowExpander（#2），用它来展示一些影片的额外信息，这些信息由于太大而无法在列中展现。

下一个重点代码是language_id列的renderer渲染器代码（#3）。我们再一次使用renderer函数来呈现存储器中已加载的某个值。我们可在此列上使用hasOne关联（在这里，影

片表跟语言表之间是多对一关系），然而，这么做之前得多问自己一些问题：虽然Ext JS提供了这种关联关系，但存储器也提供我们需要加载的值，那么这种情况下再使用（hasOne关联）合适吗？如果这么做了，从服务器端取回的JSON响应数据就会变大，并且其中的一些数据对其他模型而言就是重复的。假设这样：所有影片的language_id值是1（英语语种），则同一个语言模型就会被加载20次（pageSize的设置值）。



假设我们考虑在影片和语言模型的language_id字段上使用HasOne关联。这种情况下，因为这种关联关系，Ext JS会自动生成一个名为getLanguage的获取方法。而我们需要这么使用renderer函数：

```
dataIndex: 'language_id',
renderer: function(value, metaData, record ){
    return record.getLanguage().get('name');
}
```

接下来是分页工具栏（#4）。我们需要指定存储器，即使用同网格面板一样的存储器。由于pageSize属性已在存储器中设置了，因此这里就不需再重复设定。

最后，设置RowExpander插件的配置项（#5）。我们需要配置一个模板使其呈现一些额外信息。在这里，我们显示影片介绍以及其他一些不适合在列中呈现的信息，如租借信息等。不幸的是，无法与关联模型一同使用RowExpander插件。

前面已创建了影片网格面板。现在来实现稍早前提到的SakilaGrid类。注意，影片数据网格里还没有带添加、编辑和删除按钮的工具栏，也没有最后修改列。因此，我们即将创建的网格超类会实现这些配置。

由于我们在app/view/sakila文件夹里创建所有的视图超类，因此，也在其中创建一个名为SakilaGrid.js的文件，代码内容为：

```
Ext.define('Packt.view.sakila.SakilaGrid', {
    extend: 'Ext.grid.Panel',
    alias: 'widget.sakilagrid',

    requires: [
        'Packt.view.toolbar.AddEditDelete' // #1
    ],

    columnLines: true,
    viewConfig: {
        stripeRows: true
    },

    dockedItems: [
```

```
            {
                xtype: 'addeditdelete' // #2
            }
        ],

        initComponent: function() {
            var me = this;

            me.columns = Ext.Array.merge(me.columns, // #3
                [{
                    xtype    : 'datecolumn',
                    text     : 'Last Update',
                    width    : 120,
                    dataIndex: 'last_update',
                    format: 'Y-m-j H:i:s',
                    filter: true
                }]
            );

            me.callParent(arguments);
        }
    });
```

上述代码中，有很重要的两点需要关注。第一是AddEditDelete工具栏（#1和#2）。只要愿意，我们可以创建一个类专门声明这个工具栏。这样的话，如果想在其他组件中使用同一工具栏，就可以重用代码。另外，有了通用的工具栏，我们就创建了一个样板，并且便于后续创建控制器（监听视图的触发事件）。第二是我们声明了最后修改列（#3）。上一章实现静态数据模块时，我们用了同样的方法。

截至当前，我们还没创建AddEditDelete工具栏，那么现在就来创建它。要创建它，需先在app/view目录中创建一个名为toolbar的子目录，我们将在其中创建应用程序的所有工具栏：

```
    Ext.define('Packt.view.toolbar.AddEditDelete', {
        extend: 'Ext.toolbar.Toolbar',
        alias: 'widget.addeditdelete',

        flex: 1,
        dock: 'top',
        items: [
            {
                xtype: 'button',
                text: 'Add',
                itemId: 'add',
                iconCls: 'add'
            },
            {
                xtype: 'button',
```

```
                text: 'Edit',
                itemId: 'edit',
                iconCls: 'edit'
            },
            {
                xtype: 'button',
                text: 'Delete',
                itemId: 'delete',
                iconCls: 'delete'
            }
        ]
    });
```

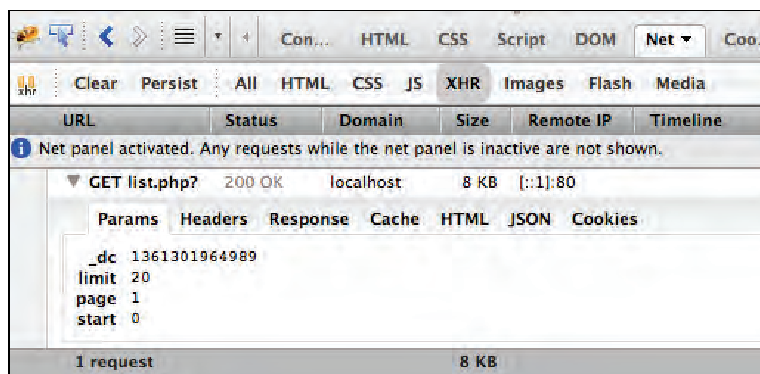
工具栏位于组件停驻项的顶部。如果我们要改变它，只需在AddEditDelete类里修改dock属性配置项即可。

请注意，我们使用的代码与前面相关类中的一样。我们可以从有关类声明中移除这些代码，并创建一个新的自定义组件以重用代码。如果你想这么做，那就去重构前面的相关类。别忘了在required声明中添加类名，否则Ext JS动态加载引擎将不知道我们想要实例化哪个类（如果它尚未在内存中加载）。

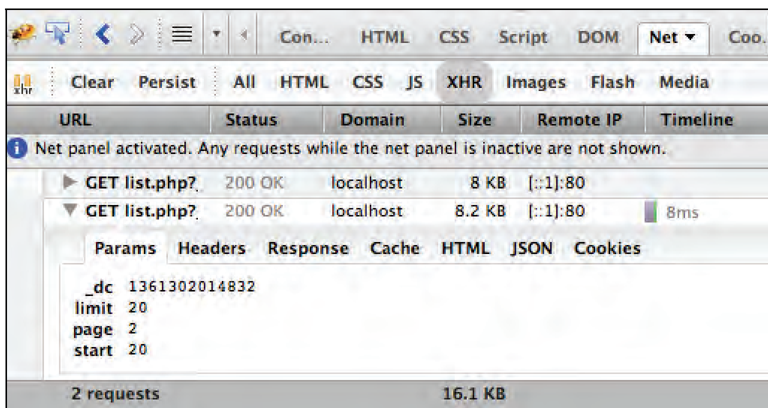
1. 处理服务器端分页

由于我们使用了分页工具栏，因此有必要记住几个概念。Ext JS提供工具帮助我们进行内容分页，但需要强调的是，如果一次性获取了数据库表的所有记录，Ext JS的分页功能将失效。看一下Ext JS发至服务器端的请求，会发现使用分页工具栏时传递了3个额外参数。

这三个参数是start、limit和page。例如，如我们所见，第一次加载网格面板信息时，start为0，limit为存储器里设置的pageSize值（这里是20），page为1：



我们点击网格面板的下一页时，start变为20，limit仍为20（这个数总是20，除非我们动态更改pageSize设置值），page变为2：





根据用户需要，已有第三方插件可以动态更改pageSize设置值：https://github.com/loiane/extjs4-ux-paging-toolbar-resizer。

这些参数同样有助我们对数据库信息进行分页。例如，对于MySQL，我们只需要start和limit，因此，可以从请求中获取它们：

```
$start = $_REQUEST['start'];
$limit = $_REQUEST['limit'];
```

之后，执行SELECT查询语句，我们需要在最后添加LIMIT $start, $limit（如果存在WHERE、ORDER BY和GROUP BY等子句，就在这些字句之后）：

```
$sql = "SELECT * FROM Film LIMIT $start,  $limit";
```

这样就能够从数据库中获取我们需要的信息。另一个很重要的细节是分页工具栏呈现数据库表记录的总数值：

```
$sql = "SELECT count(*) as num FROM Film";
```

因此我们需要在JSON格式数据中添加返回数据库表记录计数的total属性：

```
echo json_encode(array(
  "success" => $mysqli->connect_errno == 0,
  "data" => $result,
  "total" => $total
));
```

之后，Ext JS就会取回所有需要的信息并正确分页。

2. MySQL、Oracle与Microsoft SQL Server上的分页查询

如果使用不同数据库产品就需要多加小心，因为对数据库信息进行分页的查询语句是不同的。

如果使用Oracle数据库，带分页功能的SELECT查询语句就是这样：

```
SELECT * FROM
  (select rownum as rn, f.* from
    (select * from Film order by film_id) as f
  ) WHERE rn > $start  and rn <= ($start + $limit)
```

这可比MySQL的实现复杂多了。现在我们来看看Microsoft SQL Server（2012之前版本）的查询语句：

```
SELECT  *
FROM ( SELECT ROW_NUMBER() OVER ( ORDER BY film_id ) AS RowNum, *
          FROM Films
        ) AS RowConstrainedResult
WHERE   RowNum > $start
    AND RowNum <= ($start + $limit)
ORDER BY RowNum
```

SQL Server 2012的查询语句会简单点：

```
SELECT * FROM Film
ORDER BY film_id
OFFSET $start ROWS
FETCH NEXT $limit ROWS ONLY
```

Firebird数据库的的查询语句甚至比MySQL还简单：

```
SELECT FIRST $limit SKIP $start * FROM Film
```

因此，如果你使用的不是MySQL数据库，一定要注意不同的SQL实现语法。

7.2.4 创建控制器

现在我们已实现了影片网格面板。按照既定的方案，接下来要实现控制器Packt.controller.film.Films：

```
Ext.define('Packt.controller.cms.Films', {
    extend: 'Ext.app.Controller',

    requires: [  // #1
        'Packt.util.MD5',
        'Packt.util.Alert',
        'Packt.view.MyViewport',
        'Packt.util.Util'
    ],

    views: [
        'film.Films'
    ],

    stores: [
```

```
            'film.Films'
        ],

        init: function(application) {
            this.control({
                "filmsgrid": {
                    render: this.render // #2
                }
            });
        },

        render: function(component, options) {
            component.getStore().load();     // #3
        }
    });
```

以上就是我们初步实现的控制器，能够监听影片网格面板触发的render事件（#2）。影片网格面板渲染展示时，Ext JS将加载存储器（#3）。通常情况下，我们希望Ext JS加载前面创建的Util工具类（#1）。

7.3　影片网格面板编辑功能

现在，影片网格面板已经可以呈现和加载了，接下来我们实现添加和编辑功能。

如我们本章开头看到的界面截图，Edit（编辑）窗体有三个标签页：第一个用来编辑影片详细信息，第二个用来编辑与影片有关的类别信息，第三个用来编辑与影片有关的演员信息。现在，先来实现影片详细信息编辑功能。

在app/view/film文件夹中创建新的视图类Packt.view.film.FilmWindow。这个类是一个窗体，包含一个以标签面板为子项的表单。我们会在其中一个标签面板内添加表单字段，用来编辑影片详细信息。

```
Ext.define('Packt.view.film.FilmWindow', {
    extend: 'Packt.view.sakila.WindowForm', // #1
    alias: 'widget.filmwindow',

    requires: [
        'Packt.util.Util' // #2
    ],

    width: 537,
    title: 'Edit Film',
    iconCls: 'film_add',

    items: [
        {
            xtype: 'form',
            autoScroll: true,
```

```
                    layout: {
                        type: 'fit'
                    },
                    items: [
                        {
                            xtype: 'tabpanel',
                            activeTab: 0,
                            items: [
                                {
                                    xtype: 'panel',
                                    autoScroll: true,
                                    bodyPadding: 10,
                                    layout: {
                                        type: 'anchor'
                                    },
                                    title: 'Film Information',
                                    defaults: {
                                        anchor: '100%',
                                        msgTarget: 'side'
                                    },
                                    items: [ // #3
                                        //影片具体字段
                                    ]
                                },
                                //影片类别标签面板
                                //影片演员标签面板
                            ]
                        }
                    ]
                }
            ]
        });
```


现在我们有了基本配置。很重要的一点是：FilmWindow类扩展自Packt.view.sakila.WindowForm（#1），而Packt.view.sakila.WindowForm类扩展自Ext JS窗体类（Ext.window.Window）并带有Save和Cancel按钮。再次的，我们试着创建一个超类，以便尽可能重用更多代码。我们将在下一节实现这个类。

requires里声明Util类是为了在所有必填字段里使用红星标识“*”（#2）。现在，在第一个标签面板上（#3）放置表示影片数据库表每列的表单字段。

我们看一下Sakila文档里关于影片表字段的描述（http://dev.mysql.com/doc/sakila/en/sakila-structure-tables-film.html）。

- **film_id** 表的主键，具有唯一值。因此我们可以用隐藏字段控制它。
- **title** 影片标题。因此我们可以用文本字段表示它。由于这个值的最大长度是255字符，所以我们也需要加上相关的验证。
- **description** 一段影片的简短描述或情节摘要。由于描述最大长度是5000字符，因此可以用多行文本字段表示它。

- ❑ **release_year** 影片发行年份。可以用数值字段表示，从最小值1950到当前年份加1（比如说我们想添加下一年发行的影片）。
- ❑ **language_id** 指向语言表的外键，标识影片语言。可以用语言存储器驱动的组合框表示（应用程序加载时就填充好了选项值）。
- ❑ **original_language_id** 指向语言表的外键，标识影片的原产语言，用于翻译片。这个字段同样能够用语言存储器驱动的组合框表示（同样的，应用加载时就填充好了选项值）。
- ❑ **rental_duration** 租借周期的长度，以天为单位。可以用数值字段表示，最小值为1，最大值为10（限制最大值）。
- ❑ **rental_rate** 在rental_duration列的周期内，租借影片的费用。用数值字段表示，值范围从0到5，允许小数位。
- ❑ **length** 影片播放时长，以分钟为单位。用数值字段表示，值范围从1到999。
- ❑ **replacement_cost** 影片未归还或归还时有损坏的情况下客户的赔偿金。用数值字段表示，数值范围从0到100。
- ❑ **rating** 影片的评级。可以是以下某一值：G、PG、PG-13、R或NC-17。由于是固定值，可以用单选按钮或组合框表示，这里我们使用组合框。
- ❑ **special_features** 列出DVD包含的常见特性。可以是0个或多个特性，如：预告片、评论、删减内容或幕后花絮等。由于可选特性为0个或多个，因此可以用复选按钮或多选组合框表示。这里我们使用复选按钮。

我们首先声明的前3个字段是film_id、title和release_year：

```
{
    xtype: 'hiddenfield',
    name: 'film_id'
},
{
    xtype: 'textfield',
    name: 'title',
    fieldLabel: 'Title',
    afterLabelTextTpl: Packt.util.Util.required,
    allowBlank: false,
    maxLength: 255
},
{
    xtype: 'numberfield',
    name: 'release_year',
    fieldLabel: 'Release Year',
    maxValue: (new Date().getFullYear()) + 1,
    minValue: 1950,
    allowDecimals: false
}
```

目前没什么特别之处，接下来是语言字段：

```
{
    xtype: 'combobox',
    name: 'language_id',
    fieldLabel: 'Language',
    displayField: 'name',
    valueField: 'language_id',
    queryMode: 'local',
    store: 'staticData.Languages',
    afterLabelTextTpl: Packt.util.Util.required,
    allowBlank: false
},
{
    xtype: 'combobox',
    name: 'original_language_id',
    fieldLabel: 'Original Language',
    displayField: 'name',
    valueField: 'language_id',
    queryMode: 'local',
    store: 'staticData.Languages'
}
```

注意，我们为这两个字段使用了同一个存储器；我们希望它们有同样的值，这意味着当用户在静态数据模块的语言网格面板添加或改变语言时，我们希望变化能够同时反映在对应存储器上（并且这里的两个字段也能保持最新状态），所以，这里使用的存储器跟静态数据模块使用的是同一个。

数值字段有`rental_duration`、`rental_rate`、`length`和`replacement_cost`：


```
{
    xtype: 'numberfield',
    name: 'rental_duration',
    fieldLabel: 'Rental Duration',
    maxValue: 10,
    minValue: 1,
    allowDecimals: false,
    afterLabelTextTpl: Packt.util.Util.required,
    allowBlank: false
},
{
    xtype: 'numberfield',
    name: 'rental_rate',
    fieldLabel: 'Rental Rate',
    maxValue: 5,
    minValue: 0,
    step: 0.1,
    afterLabelTextTpl: Packt.util.Util.required,
    allowBlank: false
},
{
    xtype: 'numberfield',
    name: 'length',
    fieldLabel: 'Lenght (min)',
```

```
        maxValue: 999,
        minValue: 1
    },
    {
        xtype: 'numberfield',
        name: 'replacement_cost',
        fieldLabel: 'Replacement Cost',
        maxValue: 100,
        minValue: 0,
        step: 0.1,
        afterLabelTextTpl: Packt.util.Util.required
    }
```

有一点很重要，只要是数值字段并且我们想要从模型中加载它们，就需要模型中对应的字段同样是数值型（int或float，整型或浮点型），否则，表单就不会加载这些值。

rating组合框及其存储器声明如下：

```
    {
        xtype: 'combobox',
        name: 'rating',
        fieldLabel: 'Rating',
        displayField: 'text',
        valueField: 'text',
        queryMode: 'local',
        store: 'film.Ratings'
    }
```

评级的值是固定的。因此，我们可以创建一个带有预设值且扩展自ArrayStore的存储器Packt.store.film.Ratings：

```
Ext.define('Packt.store.film.Ratings', {
    extend: 'Ext.data.ArrayStore',

    fields: [
        {name: 'text'},
    ],

    data : [ // ENUM('G','PG','PG-13','R','NC-17')
        ['G'],
        ['PG'],
        ['PG-13'],
        ['R'],
        ['NC-17']
    ],

    autoLoad: true
});
```

这是我们所能创建的用以填充组合框的最简单的存储器。继续往下，special_features复选按钮组代码如下：

```
{
    xtype: 'checkboxgroup',
    fieldLabel: 'Special Features',
    columns: 2,
    name: 'special_features',
    items: [
        {
            xtype: 'checkboxfield',
            boxLabel: 'Trailers',
            inputValue: 'Trailers',
            name: 'trailers'
        },
        {
            xtype: 'checkboxfield',
            boxLabel: 'Commentaries',
            inputValue: 'Commentaries',
            name: 'commentaries'
        },
        {
            xtype: 'checkboxfield',
            boxLabel: 'Deleted Scenes',
            inputValue: 'Deleted Scenes',
            name: 'deleted'
        },
        {
            xtype: 'checkboxfield',
            boxLabel: 'Behind the Scenes',
            inputValue: 'Behind the Scenes',
            name: 'behind'
        }
    ]
}
```


复选按钮组是表单中填充起来最复杂的字段。每个复选按钮字段的行为都像一个独立的字段，因此，每个复选按钮都需要有自己的名称和输入值。我们将在开始实现控制器时介绍如何填充它。

最后，description字段是一个多行文本字段：

```
{
    xtype: 'textareafield',
    name: 'description',
    fieldLabel: 'Description',
    maxLength: 5000
}
```

7.3.1　Packt.view.sakila.WindowForm

Edit Film（影片编辑）窗体模块最后需要完成的代码段是WindowForm超类。截至目前，我们实现的所有窗体都采用Fit布局，并且一般里面都有一个表单面板。同时，窗体分别有一个

Cancel和Save按钮。由于这些配置是窗体的默认通用配置，因此我们可以考虑创建一个窗体超类：

```
Ext.define('Packt.view.sakila.WindowForm', {
    extend: 'Ext.window.Window',
    alias: 'widget.windowform',

    requires: [
        'Packt.view.toolbar.CancelSave'
    ],

    height: 400,
    width: 550,
    autoScroll: true,
    layout: {
        type: 'fit'
    },
    modal: true,

    //子类中必须覆盖的项

    dockedItems: [
        {
            xtype: 'cancelsave'
        }
    ]
});
```

我们可以按类似思路对Cancel Save Toolbar（取消保存工具栏）做同样处理：

```
Ext.define('Packt.view.toolbar.CancelSave', {
    extend: 'Ext.toolbar.Toolbar',
    alias: 'widget.cancelsave',

    flex: 1,
    dock: 'bottom',
    ui: 'footer',
    layout: {
        pack: 'end',
        type: 'hbox'
    },
    items: [
        {
            xtype: 'button',
            text: 'Cancel',
            itemId: 'cancel',
            iconCls: 'cancel'
        },
        {
            xtype: 'button',
            text: 'Save',
            itemId: 'save',
            iconCls: 'save'
        }
    ]
});
```

我们又一次游刃有余地重构了我们已实现的代码。这就是Ext JS和MVC架构带来的好处：允许你重用代码，并且如同在其他面向对象语言里可以做的那样，你可以重构这些代码，而且实现起来并不难。

7.3.2 影片类别

前面我们已经介绍了影片的详细资料，接下来我们处理最复杂的部分：film表与category表和actor表的关联关系。category表和actor表都跟film表有一个多对多的关系。在开始对关联关系进行编码之前，我们需要再次思考一下：是否真的需要一个关联关系？值得吗？这些关联关系是否会造成服务器端与Ext JS客户端之间的数据交换发生数据过载现象呢？

看看数据库，就会发现每部影片只有一个类别，尽管存在多对多关系。即使Ext JS具备管理关联关系的能力，我们也不打算马上使用它。因为我们希望只在用户打开编辑窗体浏览影片信息时才加载关联信息，即按需加载关联数据。

但是，如果真要建立关联，那我们怎样在Ext JS里处理一个多对多关系呢？Ext JS并没有提供原生支持。你可以在film表与film_category表之间建立一个名为FilmCategory的Has Many关联（这里是一对多关联关系），然后在FilmCategory跟category表间建立Has One关联（这里是多对一关联关系）。

1. Store

由于我们只打算显示与某部具体影片关联的类别，因此可以重用Category模型，但需要创建一个新的存储器：

```
Ext.define('Packt.store.film.FilmCategories', {
    extend: 'Ext.data.Store',

    requires: [
        'Packt.model.staticData.Category',
        'Packt.proxy.Sakila'
    ],

    model: 'Packt.model.staticData.Category',

    proxy: {
        type: 'sakila',

        api: {
            create: 'php/inventory/film_category_create.php',
            read: 'php/inventory/film_category.php',
            update: 'php/inventory/film_category_update.php',
            destroy: 'php/inventory/film_category_destroy.php'
        }
    }
});
```

我们将通过这个存储器执行film_category数据库表上的CRUD操作。但现在我们首先关注read动作。

我们的思路是传递感兴趣的film_id到categorie表，然后将其发送到服务器端。通过以下查询语句可从categorie表里获取我们需要的信息：

```
SELECT c.category_id, c.name, f.last_update FROM category c
INNER JOIN film_category f ON f.category_id= c.category_id
WHERE f.film_id= $film_id
```

这样我们就不需要使用关联关系，并且可以只通过一条SELECT语句获取我们需要的信息。

2. 编辑视图

下一步是实现Edit Film（影片编辑）窗体上网格面板显示影片类别功能：

```
Ext.define('Packt.view.film.FilmCategories', {
    extend: 'Packt.view.sakila.SakilaGrid',
    alias: 'widget.filmcategories',

    requires: [
        'Packt.view.toolbar.SearchAddDelete'
    ],

    store: 'film.FilmCategories',

    columns: [
        {
            text: 'Category Id',
            width: 100,
            dataIndex: 'category_id'
        },
        {
            text: 'Category Name',
            flex: 1,
            dataIndex: 'name'
        }
    ],

    dockedItems: [
        {
            xtype: 'searchadddelete' // #1
        }
    ]
});
```

这是一个简单的网格面板，跟之前创建的网格面板很相似，但其工具栏（#1）上的按钮是搜索、添加、删除等，而非添加、编辑和删除。这又是一个可以说明我们能够覆写超类配置（SakilaGrid，初始配置中已有了一个带添加、编辑、删除按钮的工具栏）的好例子。如果不覆写超类配置项，那么就将使用SakilaGrid类中原来声明的属性配置项。由于我们覆写了

dockedItem配置项，所以这里将使用子类中声明的配置项。

3. 带搜索、添加、删除按钮的工具栏

这个自定义工具栏组件于我们而言是全新的，之前从未遇到。由于Actor（演员）界面也使用Search（搜索）、Add和Delete（删除）按钮，因此，我们专门创建一个工具栏组件类：

```
Ext.define('Packt.view.toolbar.SearchAddDelete', {
    extend: 'Ext.toolbar.Toolbar',
    alias: 'widget.searchadddelete',

    flex: 1,
    dock: 'top',
    items: [
        {
            xtype: 'button',
            text: 'Search and Add',
            itemId: 'add',
            iconCls: 'find'
        },
        {
            xtype: 'button',
            text: 'Delete',
            itemId: 'delete',
            iconCls: 'delete'
        }
    ]
});
```

这是一个带有两个按钮的简单工具栏，跟我们之前实现的工具栏很相似，如下图所示：



4. 搜索类别——**MultiSelect**组件

用户点击Search and Add（搜索和添加）按钮时，将呈现一个新窗体，其中包含一个MultiSelect（多选）组件，用户可以选择1个或多个类别添加到影片类别中：

```
Ext.define('Packt.view.film.SearchCategory', {
    extend: 'Packt.view.sakila.SearchWindow',
    alias: 'widget.searchcategory',

    requires: [
        'Ext.ux.form.MultiSelect'
    ],

    title: 'Add Category',
```

```
    items: [
        {
            xtype: 'form',
            itemId: 'filmForm',
            autoScroll: true,
            bodyPadding: 10,
            items: [
                {
                    xtype: 'label', // #1
                    text: 'Hold Ctrl or Command to select more than one Category.'
                },
                {
                    anchor: '100%',
                    xtype: 'multiselect', // #2
                    msgTarget: 'side',
                    fieldLabel: 'Categories',
                    name: 'multiselect',
                    allowBlank: false,
                    store: 'staticData.Categories', // #3
                    valueField: 'category_id', // #4
                    displayField: 'name',      // #5
                    ddReorder: true            // #6
                }
            ]
        }
    ]
});
```

我们有一个表单，表单面板上有两个子项：第一个（#1）是标签项，通知用户按下Ctrl或Command键选择多个类别；第二个是`MultiSelect`字段（#2），这个字段并非是Ext JS原生自带的，它位于示例目录下的ux文件夹中，因此，我们将在`requires`属性配置里添加该类的声明。

`MultiSelect`组件与组合框组件很相似。你也需要设置一个存储器（#3）、`valueField`（#4）以及`displayField`（#5）。不同点在于信息呈现给用户的方式。区别于下拉列表，它是一个带有多选值的面板。

为了更有趣一些，我们将实现拖放重排，用户可以通过拖放来对值进行重新排序（#6）。

5. `Packt.view.sakila.SearchWindow`

由于`SearchCategory`类扩展自`SearchWindow`，因此需要创建`SearchWindow`类。这同样是一个初次遇到的新组件。如果后面我们还需要创建Search and Add窗体，那么就可以扩展自这个类：

```
Ext.define('Packt.view.sakila.SearchWindow', {
    extend: 'Ext.window.Window',
    alias: 'widget.searchWindow',

    requires: [
        'Packt.view.toolbar.CancelClearAdd'
```

```
    ],

    height: 300,
    width: 400,
    autoScroll: true,
    layout: {
        type: 'fit'
    },
    iconCls: 'find',
    modal: true,

    //子类必须覆盖的项

    dockedItems: [
        {
            xtype: 'cancelclearadd'
        }
    ]
});
```

我们发现，它跟编辑窗体很相似，不同点在于我们使用了带有Cancel、Clear（清除）和Add按钮的工具栏，而非取消和保存按钮。

带有Cancel、Clear和Add按钮的工具栏同样非常简单（跟我们目前已实现的工具栏没什么区别）：

```
Ext.define('Packt.view.toolbar.CancelClearAdd', {
    extend: 'Ext.toolbar.Toolbar',
    alias: 'widget.cancelclearadd',

    flex: 1,
    dock: 'bottom',
    ui: 'footer',
    layout: {
        pack: 'end',
        type: 'hbox'
    },
    items: [
        {
            xtype: 'button',
            text: 'Cancel',
            itemId: 'cancel',
            iconCls: 'cancel'
        },
        {
            xtype: 'button',
            text: 'Clear',
            itemId: 'clear',
            iconCls: 'clear'
        },
        {
            xtype: 'button',
```

```
            text: 'Add Selected',
            itemId: 'save',
            iconCls: 'save'
        }
    ]
});
```

后续，我们将在控制器中处理这些按钮触发的所有事件。

7.3.3 演员信息

actor表与film表的关联关系，类似于category表与film表的关系，也是一个多对多的关系。我们按照处理影片表和类别表关系的思路来处理演员表与影片表的多对多关联关系。

1. 存储器

我们可以再次重用Actor模型，只需简单地创建一个新的存储器来处理关联关系：

```
Ext.define('Packt.store.film.FilmActors', {
    extend: 'Ext.data.Store',

    requires: [
        'Packt.model.staticData.Actor',
        'Packt.proxy.Sakila'
    ],

    model: 'Packt.model.staticData.Actor',

    proxy: {
        type: 'sakila',

        api: {
            create: 'php/inventory/film_actor_create.php',
            read: 'php/inventory/film_actor.php',
            update: 'php/inventory/film_actor_create.php',
            destroy: 'php/inventory/film_actor_delete.php'
        }
    }
});
```

在服务器端，我们将像处理film_category表那样：发送film_id到服务器端，然后从actor表中取回我们感兴趣的信息：

```
SELECT c.actor_id, c.first_name, c.last_name, f.last_update
FROM actor c
INNER JOIN film_actor f ON f.actor_id= c.actor_id
WHERE f.film_id= $film_id
```

我们保存一部影片的编辑内容时，film_actor表将被修改或者添加新记录。

2. 编辑视图

与处理film_category表的思路类似：我们也需要一个网格面板显示演员信息。再一次扩展SakilaGrid类并显示演员表的相关列：Actor Id、First Name以及Last Name（最后修改列也会被显示，因为已经在SakilaGrid类里实现了它）：

```
Ext.define('Packt.view.film.FilmActors', {
    extend: 'Packt.view.sakila.SakilaGrid',
    alias: 'widget.filmactors',

    requires: [
        'Packt.view.toolbar.SearchAddDelete'
    ],

    store: 'film.FilmActors',

    columns: [
        {
            text: 'Actor Id',
            width: 100,
            dataIndex: 'actor_id'
        },
        {
            text: 'First Name',
            flex: 1,
            dataIndex: 'first_name'
        },
        {
            text: 'Last Name',
            width: 200,
            dataIndex: 'last_name'
        }
    ],

    dockedItems: [
        {
            xtype: 'searchadddelete'
        }
    ]
});
```

这个网格面板上有Search and Add（搜索和添加）、Delete（删除）按钮。

3. 搜索演员表——即席搜索组合框

即席搜索组合框的思路是呈现搜索界面以及一个组合框字段给用户，用户可以输入搜索条件，之后系统进行即席搜索并显示匹配用户搜索条件的演员信息以及演员参演的影片信息。所有与搜索条件相匹配的演员都将作为组合框的选择项显示出来，并且组合框可分页展示。用户选择演员时，我们将显示演员的姓和名。这是个非常棒的组件，也比其他的组件更复杂，我们将为组合框应用一些高级配置。

4. 模型

首先，需要一个模型来呈现从服务器端获取的信息。我们将获取演员及其参演影片的信息。因此，可以创建一个扩展自Actor模型的SearchActor模型，在其中只需声明缺失的字段即可：

```
Ext.define('Packt.model.film.SearchActor', {
    extend: 'Packt.model.staticData.Actor',

    fields: [
        { name: 'film_info' }
    ]
});
```

5. 存储器

接下来我们需要实现一个存储器，用来加载SearchActor模型集：

```
Ext.define('Packt.store.film.SearchActors', {
    extend: 'Ext.data.Store',

    requires: [
        'Packt.model.film.SearchActor'
    ],

    model: 'Packt.model.film.SearchActor',

    pageSize: 2,

    proxy: {
        type: 'ajax',
        url: 'php/inventory/searchActors.php',
        reader: {
            type: 'json',
            root: 'data'
        }
    }
});
```

在服务器端，我们将通过actor_info数据库视图获取信息。另外，组合框也传递了三个额外的参数：用于分页的start和limit参数，以及一个包含用户所输搜索条件、名为query的参数。

因此，我们的SELECT查询语句如下：

```
$start = $_REQUEST['start'];
$limit = $_REQUEST['limit'];
$query = $_REQUEST['query'];

//查询并获取信息
$sql = "SELECT * FROM actor_info ";
$sql .= "WHERE first_name LIKE '%" . $query . "%' OR ";
$sql .= "last_name LIKE '%" . $query . "%' ";
$sql .= "LIMIT $start,  $limit";
```

当我们实现分页功能时，别忘了统计一下共有多少条匹配搜索条件的记录，并通过JSON格式数据的`total`属性返回给客户端：

```
$sql = "SELECT count(*) as num FROM actor_info ";
$sql .= "WHERE first_name LIKE '%" . $query . "%' OR ";
$sql .= "last_name LIKE '%" . $query . "%' ";
```

现在，我们就可以根据用户输入的搜索条件获取数据库信息了。

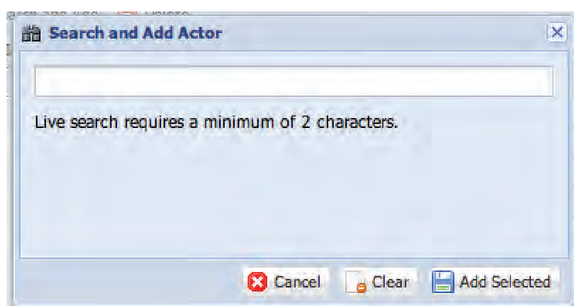
6. 即席搜索组合框

接下来，我们要实现提供搜索功能的视图。因此，创建一个扩展自`SearchWindow`的类，其中包含一个组合框，它能提供即席搜索的所有功能。

```
Ext.define('Packt.view.film.SearchActor', {
    extend: 'Packt.view.sakila.SearchWindow',
    alias: 'widget.searchactor',
    width: 600,
    bodyPadding: 10,
    layout: {
        type: 'anchor'
    },
    title: 'Search and Add Actor',

    items: [
        {
            //组合框 // #1
        },{
            xtype: 'component',
            style: 'margin-top:10px',
            html: 'Live search requires a minimum of 2 characters.'
        }
    ]
});
```

搜索框底部有一个用户提示：至少输入两个字符才能进行即席搜索。如下图所示：



现在，我们来实现上述代码中#1位置的组合框功能：

7

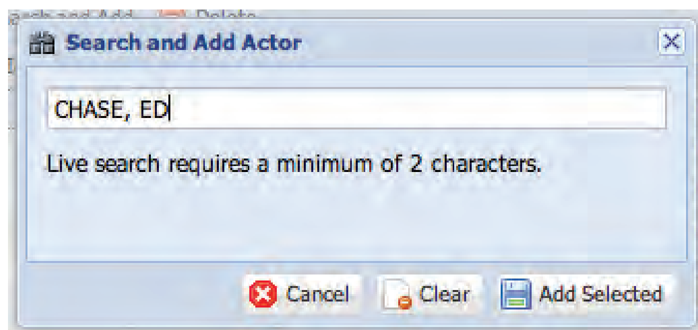
```
xtype: 'combo',
store: 'film.SearchActors', // #1
displayField: 'first_name', // #2
valueField: 'actor_id',     // #3
typeAhead: false,
hideLabel: true,
hideTrigger:true,   // #4
anchor: '100%',
minChars: 2,       // #5
pageSize: 2,       // #6

displayTpl: new Ext.XTemplate(       // #7
    '<tpl for=".">' +
        '{[typeof values === "string" ? values : values["last_name"]]}, ' +
        '{[typeof values === "string" ? values : values["first_name"]]}' +
    '</tpl>'
),

listConfig: {         // #8
    loadingText: 'Searching...',
    emptyText: 'No matching posts found.',

    // 为每个选项自定义渲染模板
    getInnerTpl: function() {         // #9
        return '<h3><span>{last_name}, {first_name}</span></h3></br>' +
            '{film_info}';
    }
}
```

同样的，我们需要一个存储器（#1，前面已经实现了它）用来填充组合框。然后，我们需要一个displayField属性配置项（#2）。当即席搜索选中一个演员时，displayField将只显示演员表的first_name字段值。然而，我们希望能够显示last_name和first_name两个字段值。因此，为了达成这个目的，我们需要覆写displayTpl模板（#7）。效果如下图所示：



接下来，设置valueField属性配置项（#3）为所选演员的id；通过设置hideTrigger属性配置项隐藏下拉箭头（#4）；即席搜索操作至少需要用户输入两个字符才能工作（#5）；组合框每页只显示两个演员（#6）。

然后，设置listConfig属性配置项（#8），我们可以在这里配置加载提示信息、无值提示信息以及显示演员信息的模板等。一般情况下，显示演员信息时，开头加粗显示演员的last_name和first_name信息，下一行显示演员参演的所有影片。

7.4　影片控制器

通过前面几章学习，我们掌握了数据保存的实现方法：表单提交、Ajax请求以及通过存储器写入。本节我们将只聚焦尚未实现的功能。但不用担心，完整实现的源代码都能在本书的对应章节找到。

7.4.1　在编辑表单中加载已有影片信息

我们已经尝试过通过loadRecord方法加载表单面板（在第5章）。现在来试试另一种方法：setValues方法。因为我们有个CheckBox（复选按钮）组，在表单面板上加载它的值还是有点复杂的，但这个方法比较适合当前情形。当用户从影片网格面板中选择了一条记录并点击Edit按钮时，将打开编辑窗体并加载记录值。基于上述思路，我们需要监听Edit按钮的click事件并实现一个以button（按钮）为参数的方法：

```
var grid= button.up('filmsgrid'),
record = grid.getSelectionModel().getSelection();

if(record[0]){ // #1

    var editWindow = Ext.create('Packt.view.film.FilmWindow');

    var form = editWindow.down('form');

    var values = { // #2
        film_id: record[0].get('film_id'),
        title: record[0].get('title'),
        description: record[0].get('description'),
        release_year: record[0].get('release_year'),
        language_id: record[0].get('language_id'),
        original_language_id: record[0].get('original_language_id'),
        rental_duration: record[0].get('rental_duration'),
        rental_rate: record[0].get('rental_rate'),
        length: record[0].get('length'),
        replacement_cost: record[0].get('replacement_cost'),
        rating: record[0].get('rating')
    };

    Ext.each(record[0].get('special_features').split(','), function(feat){ // #3
        if (feat === 'Trailers') {
            values.trailers = 'Trailers';
        } else if (feat === 'Commentaries') {
            values.commentaries = 'Commentaries';
```

7

```
        } else if (feat === 'Deleted Scenes') {
            values.deleted = 'Deleted Scenes';
        } else if (feat === 'Behind the Scenes') {
            values.behind = 'Behind the Scenes';
        }
    });

    var filmCategories = editWindow.down('form filmcategories');
    filmCategories.getStore().load({   // #4
        params: {
            filmId: record[0].get('film_id')
        }
    });

    var filmActors = editWindow.down('form filmactors');
    filmActors.getStore().load({       // #5
        params: {
            filmId: record[0].get('film_id')
        }
    });

    form.getForm().setValues(values); // #6

    editWindow.setTitle(record[0].get('title'));
    editWindow.setIconCls('film_edit'); // #7
    editWindow.show();
}
```

首先，如果用户从网格面板选择了一条记录（#1），我们将创建编辑窗体，获取表单面板引用，取出记录值（#2）。其中，如果在模型中该字段未设置正确类型，就需要做转换（数值字段只能接受数字，无法接受字符串）。

但上述代码还缺少`special_features`字段，数据库表中该字段值是用逗号分割的文本信息，我们需要获取每个分割值（#3）。每个复选按钮看起来都像一个个独立字段，因此，我们需要给每个需要设置的复选按钮字段设置输入值。

接下来，我们获取`film_id`字段值并加载`FilmCategories`（#4）和`FilmActors`（#5）存储器，但只能获取`film_id`字段值对应影片的关联信息。

当一切准备就绪后，我们调用`setValues`方法，并传入前面创建的JSON对象（#6），该JSON对象包装了所有需设置的表单字段的值。

接下来，我们动态设置窗体的`Title`和`Icon`属性值（#7），并把窗体呈现给用户。

7.4.2　获取MultiSelect组件值

一旦用户选择了选定影片关联的正确类别，就可以点击保存按钮。这时我们需要处理按钮的点击事件，处理逻辑如下：

```
var searchWindow = button.up('searchcategory');
var values = searchWindow.down('multiselect').getValue();
var store = Ext.getStore('categories');
var filmCategoriesStore = this.getFilmCategories().getStore();

Ext.each(values, function(value){

   var model = store.findRecord('category_id', value);

      if (model){
            model.set('last_update', new Date());
            filmCategoriesStore.add(model);
      }
});

searchWindow.close();
```

首先获取SearchCategory的引用，然后定位到multiselect字段并获取值。由于multiselect字段只返回所选项对应的id（即获取到的值是id），因此，接着需要获取Categories存储器的引用，后续用来查找并获取id对应的类别。最后，我们同样需要获取FilmCategories存储器的引用，用以添加id对应的类别。

接下来，我们在Categories存储器中对每个值都进行查找，如果找到了对应模型，就添加该模型到FilmCategories存储器中，该模型将在FilmCategories网格面板中显示。

但是，要想正常工作，还需要this.getFilmCategories方法运行起来。我们创建一个FilmCategories网格面板引用来解决这个问题：

```
{
      ref: 'filmCategories',
      selector: 'filmcategories'
}
```

7.4.3　通过即席搜索获取所选演员

要获取即席搜索的演员的详细信息，我们可以采取处理影片分类时所用的方式。由于用到了不同组件，所以，相应的实现也就有所不同：

```
var searchWindow = button.up('searchactor');
var value = searchWindow.down('combo').getValue();
var store = Ext.getStore('actors');
var model = store.findRecord('actor_id', value);

if (model){
      model.set('last_update', new Date());
      this.getFilmActors().getStore().add(model);
}

searchWindow.close();
```

首先获取SearchActor的引用，然后定位到组合框字段并获取所选值。由于组合框字段只返回所选项对应的id（即获取到的值是id），因此，接着需要获取Actors存储器的引用，用来查找演员对应的模型。最后，我们同样需要获取FilmActors存储器的引用，用以添加对应的模型。

然后，搜索Actors存储器，如果找到对应的模型，我们就将其添加到FilmActors存储器中，对应的模型将在FilmActors网格面板中显示。

同样的，要想正常工作，还需要this.getFilmActors方法运行起来，我们创建一个filmActors网格面板引用来解决这个问题：

```
{
      ref: ' filmActors',
      selector: ' filmactors'
}
```

7.5　小结

本章我们了解了怎样实现一个更复杂的界面，用来管理数据库表中的库存信息。同时，我们学习了两种处理多对多关联的方式。

我们还学习了MultiSelect组件的使用方法，以及如何通过表单和组合框组件完成即席搜索功能。

下一章我们将学习如何在已开发的界面中添加一些非原生Ext JS API提供的额外功能，如对网格面板内容进行打印、导出Excel或PDF格式等。我们还将学习如何实现图表功能，并将其导出为图片和PDF格式。

第 8 章

添加额外功能

8

我们应用程序的开发已经到了收尾阶段，而且Ext JS提供了强大的功能，但仍有一些功能需要借助其他技术自己编码实现。虽说现在已经实现了一个具备分页、排序以及过滤功能的网格面板，但有时候用户还是希望应用程序能够提供更多的功能。比如添加打印、导出Excel，以及将图表导出成图片和PDF等功能，这可以为应用程序增色并让最终用户满意。

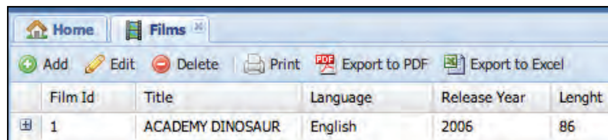
因此，本章主要包括以下内容：

- 打印网格面板上的记录；
- 将网格面板信息导出成PDF和Excel格式；
- 图表功能；
- 图表导出成PDF和图片格式；
- 使用第三方插件。

8.1 将网格面板信息导出成 PDF 和 Excel 格式

我们要实现的第一个功能是把网格面板上的内容导出成PDF和Excel格式。我们先为在上一章实现的Films（影片）网格面板增加这个功能。然而，对于其他任何Ext JS应用程序中的网格面板，实现此功能的逻辑思路都是一样的。

我们要做的第一件事就是在网格面板工具栏上添加导出按钮。我们将添加3个按钮：一个是Print（打印）网格面板内容（后续会开发这个功能，但现在先添加它），一个是Export to PDF（导出成PDF），再一个是Export to Excel（导出成Excel）。



还记得上一章我们为网格面板创建了一个AddEditDelete工具栏吗？我们准备在它上面添加这3个按钮：

```
Ext.define('Packt.view.toolbar.AddEditDelete', {
    extend: 'Ext.toolbar.Toolbar',
    alias: 'widget.addeditdelete',

    items: [
        // 添加、编辑和删除按钮
        {
            xtype: 'tbseparator'
        },
        {
            xtype: 'button',
            text: 'Print',
            itemId: 'print',
            iconCls: 'print'
        },
        {
            xtype: 'button',
            text: 'Export to PDF',
            itemId: 'pdf',
            iconCls: 'pdf'
        },
        {
            xtype: 'button',
            text: 'Export to Excel',
            itemId: 'excel',
            iconCls: 'excel'
        }
    ]
});
```

别忘了为每个按钮添加itemId属性，以便我们可以在稍后实现的控制器里监听到特定按钮触发的事件。如果你尚未重构代码并添加这个工具栏到每个网格面板中，那请记得为每个网格面板添加上这些功能。

我们添加了3个按钮到工具栏，这也是重构代码并更改类名和别名的一个机会。当然，如果我们要这么做，需记得更改所有引用AddEditDelete类的代码。

8.1.1　导出成PDF格式

现在按钮已经显示在影片网格面板上了，我们需要回到影片控制器，并添加这些功能。

我们首先要监听Export to PDF按钮点击事件。用户点击这个按钮时，将执行下面代码：

```
onButtonClickPDF: function(button, e, options) {
    var mainPanel = Ext.ComponentQuery.query('mainpanel')[0];//#1

    newTab = mainPanel.add({ // #2
        xtype: 'panel',
        closable: true,
        iconCls: 'pdf',
```

```
        title: 'Films PDF',
        layout: 'fit',
        items: [{
            xtype: 'uxiframe', // #3
            src: 'php/pdf/exportFilmsPdf.php' // #4
        }]
    });

    mainPanel.setActiveTab(newTab);
}
```

我们想实现的是当用户点击Export to PDF按钮时，打开一个新的标签页，里面呈现PDF文件。这意味着我们需要获取应用系统视见区中央主面板项（#1），并在其中添加一个标签页（#2）；由于PDF文件在里面，可考虑实现一个`iFrame`。要在Ext JS中实现一个`iFrame`，可以使用随SDK发布的`IFrame`插件（#3，在examples/ux目录下）。由于我们已经在app.js文件加载器配置项里添加了Ext.ux的映射，因此，只需简单地在`requires`声明里请求这个插件就行了。这样，当我们加载控制器时，这个插件将被加载，当Ext JS尝试用它的`xtype`（#3）进行实例化时，插件就已被加载了：

```
requires: [
    //这里放其他声明
    'Ext.ux.IFrame'
]
```

下面到了最重要的部分：Ext JS并未提供原生的导出成PDF格式的功能。如果我们想在应用程序里实现这个功能，就需要使用不同的技术。在这里，PDF文件将在服务器端生成（#4），在`iFrame`里显示。

执行上面代码，将得到以下输出：

Home | Films | Films PDF

Film Id	Title	Language	Release Year	Length	Rating	Rental Duration	Rental Rate	Last Update
1	ACADEMY DINOSAUR	English	2006	86	PG	6 days	0.99	2006-02-15 05:03:42
2	ACE GOLDFINGER	English	2006	48	G	3 days	4.99	2006-02-15 05:03:42
3	ADAPTATION HOLES	English	2006	50	NC-17	7 days	2.99	2006-02-15 05:03:42
4	AFFAIR PREJUDICE	English	2006	117	G	5 days	2.99	2006-02-15 05:03:42
5	AFRICAN EGG	English	2006	130	G	6 days	2.99	2006-02-15 05:03:42
6	AGENT TRUMAN	English	2006	169	PG	3 days	2.99	2006-02-15 05:03:42
7	AIRPLANE SIERRA	English	2006	62	PG-13	6 days	4.99	2006-02-15 05:03:42
8	AIRPORT POLLOCK	English	2006	54	R	6 days	4.99	2006-02-15 05:03:42
9	ALABAMA DEVIL	English	2006	114	PG-13	3 days	2.99	2006-02-15 05:03:42
10	ALADDIN CALENDAR	English	2006	63	NC-17	6 days	4.99	2006-02-15 05:03:42
11	ALAMO VIDEOTAPE	English	2006	126	G	6 days	0.99	2006-02-15 05:03:42
12	ALASKA PHANTOM	English	2006	136	PG	6 days	0.99	2006-02-15 05:03:42
13	ALI FOREVER	English	2006	150	PG	4 days	4.99	2006-02-15 05:03:42
14	ALICE FANTASIA	English	2006	94	NC-17	6 days	0.99	2006-02-15 05:03:42
15	ALIEN CENTER	English	2006	46	NC-17	5 days	2.99	2006-02-15 05:03:42
16	ALLEY EVOLUTION	English	2006	180	NC-17	6 days	2.99	2006-02-15 05:03:42
17	ALONE TRIP	English	2006	82	R	3 days	0.99	2006-02-15 05:03:42
18	ALTER VICTORY	English	2006	57	PG-13	6 days	0.99	2006-02-15 05:03:42
19	AMADEUS HOLY	English	2006	113	PG	6 days	0.99	2006-02-15 05:03:42
20	AMELIE HELLFIGHTERS	English	2006	79	R	4 days	4.99	2006-02-15 05:03:42
21	AMERICAN CIRCUS	English	2006	129	R	3 days	4.99	2006-02-15 05:03:42
22	AMISTAD MIDSUMMER	English	2006	85	G	6 days	2.99	2006-02-15 05:03:42
23	ANACONDA CONFESSIONS	English	2006	92	R	3 days	0.99	2006-02-15 05:03:42
24	ANALYZE HOOSIERS	English	2006	181	R	6 days	2.99	2006-02-15 05:03:42
25	ANGELS LIFE	English	2006	74	G	3 days	2.99	2006-02-15 05:03:42
26	ANNIE IDENTITY	English	2006	86	G	3 days	0.99	2006-02-15 05:03:42
27	ANONYMOUS HUMAN	English	2006	179	NC-17	7 days	0.99	2006-02-15 05:03:42
28	ANTHEM LUKE	English	2006	91	PG-13	5 days	4.99	2006-02-15 05:03:42

Mastering ExtJS book – Loiane Groner – http://packtpub.com

在服务器端生成PDF文件（PHP实现）

由于我们需要在服务器端生成PDF文件，所以可以考虑采用一些服务器端语言对应的框架或类库来实现这个功能。我们选择TCPDF类（http://www.tcpdf.org/）在PHP中生成PDF文件。还有其他相似的类库同样适用，你可以选择你最熟悉的一种。



> 如果你使用Java，可以选择iText（http://itextpdf.com/），如果使用.NET，则可以选择iTextSharp（http://itextpdf.com/ ）。

8.1.2　导出成Excel格式

要把网格面板信息导出到Excel文件，同样需要通过服务器端技术来实现。我们使用PHPExcel类库来完成该功能（http://phpexcel.codeplex.com/）。

在Ext JS端，我们唯一要做的工作就是调用生成Excel文件的处理链接：

```
onButtonClickExcel: function(button, e, options) {
    window.open('php/pdf/exportFilmsExcel.php');
}
```



> 如果你使用Java，可以选择Apache的POI库（http://poi.apache.org/），如果使用.NET，则可以选择ExcelLibrary（https://code.google.com/p/excellibrary/）。

如果你希望将其他网格面板的信息导出成Excel、PDF、Text以及Word文件，可以采取同样的方法。

8.2　通过网格打印插件打印网格面板内容

接下来要实现的功能是打印网格面板的内容。当用户点击Print（打印）按钮时，应用程序将打开一个新的浏览窗体并在其中显示网格内容。

为完成这个功能，我们将使用一个名为`Ext.ux.grid.Printer`的插件，该插件接受待打印网格面板的引用，获取存储器中的信息，生成打印内容的HTML格式，之后在新窗体中显示。



网格打印插件是个第三方插件，可通过https://github.com/loiane/extjs4-ux-gridprinter下载。这个插件只打印网格面板存储器提供的可用信息，也就是说，如果你使用了分页工具栏，该插件就只生成当前页信息的HTML内容。该插件也提供了行扩展插件功能。还请随时为该插件（或其他Ext JS插件）做些免费的贡献，这样可以更好地帮助Ext JS社区发展。

安装完该插件后（从ux目录中获取副本并放至masteringextjs/extjs/ux目录下），我们只需在`Films`控制器的`requires`声明里添加该插件请求即可：

```
requires: [
    // 这里放其他声明
    'Ext.ux.grid.Printer'
]
```

当用户点击Print按钮时，控制器将执行以下方法：

```
onButtonClickPrint: function(button, e, options) {
    Ext.ux.grid.Printer.printAutomatically = false;
    Ext.ux.grid.Printer.print(button.up('filmsgrid'));
}
```

`printAutomatically`属性控制是否自动显示打印窗体。如果设为`false`，该插件将不会显示打印窗体，接下来，如果用户想要打印，需要在浏览菜单里选择Print（Ctrl+P）。

要让插件正常工作，我们需要传递网格面板的引用给`print`方法。在这里，可以用`button.up`方法获取影片网格面板的引用。

当我们执行代码时，将得到以下输出结果：

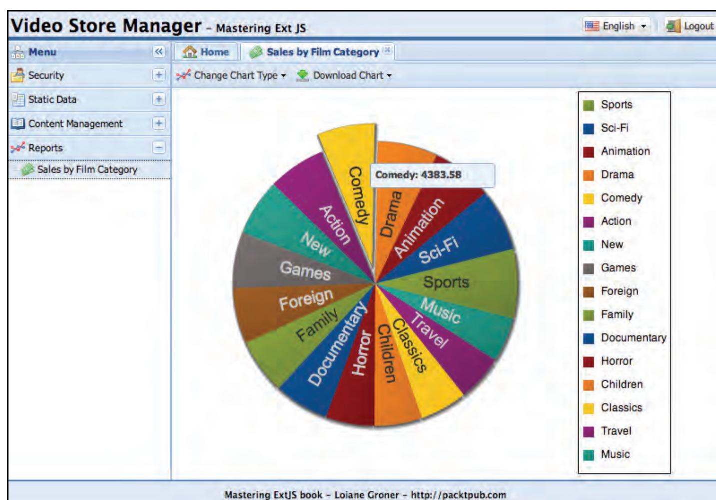
Print　Close

Film Id	Title	Language	Release Year	Lenght	Rating	Last Update
1	ACADEMY DINOSAUR	English	2006	86	PG	02/15/2006
Description: A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies **Special Features:** Deleted Scenes,Behind the Scenes **Rental Duration:** 6 **Rental Rate:** 0.99 **Replacement Cost:** 20.99						
2	ACE GOLDFINGER	English	2006	48	G	02/15/2006
Description: A Astounding Epistle of a Database Administrator And a Explorer who must Find a Car in Ancient China **Special Features:** Trailers,Deleted Scenes **Rental Duration:** 3 **Rental Rate:** 4.99 **Replacement Cost:** 12.99						
3	ADAPTATION HOLES	English	2006	50	NC-17	02/15/2006
Description: A Astounding Reflection of a Lumberjack And a Car who must Sink a Lumberjack in A Baloon Factory **Special Features:** Trailers,Deleted Scenes **Rental Duration:** 7 **Rental Rate:** 2.99 **Replacement Cost:** 18.99						
4	AFFAIR PREJUDICE	English	2006	117	G	02/15/2006

8.3　创建影片类别销售图

Ext JS提供了一整套可供使用的可视化图表，这可是令用户高兴的事。我们接下来实现三种不同的图表类型（饼图、柱状图和条形图），用户可以通过它看到不同类别影片的销售情况（Sales by Film Category）。

下图是本节完成后的最终结果的截图。从图中可看出，我们有个图表，其上有个工具栏，工具栏上有两个按钮：Change Chart Type（更改图表类型）按钮，用户可以更改图表类型（在饼图、柱状图或条形图间转换）；Download Chart（下载图表）按钮，用户能够以图片、SVG或PDF格式下载图表。



以下部分组成一个图表：store 提供数据；series 表示我们希望创建的图表类型（饼图、柱状图、条形图，等等）；axis 为坐标轴属性，如果图表是一个基于直角坐标系（也叫笛卡儿坐标系）的图表，那么它就有*X*轴和*Y*轴。

不管是创建饼图、柱状图还是条形图，我们都需要一个存储器提供图表中展现的信息。因此，先来创建一个新的存储器SalesFilmCategory：

```
Ext.define('Packt.store.reports.SalesFilmCategory', {
    extend: 'Ext.data.Store',

    requires: [
        'Packt.proxy.Sakila'
    ],
    fields: [ // #1
        {name: 'category'},
        {name: 'total_sales'}
    ],
```

```
    autoLoad: true,

    proxy: {
        type: 'sakila', // #2
        url: 'php/reports/salesFilmCategory.php'
    }
});
```

对于这个存储器，我们并未声明对应的模型，而是直接在存储器里声明了所需字段（#1）。由于这个存储器是图表专用的，因此没必要专门为其创建一个特定模型，而且我们也没打算在后面重用这个模型。

我们可以重用Sakila代理配置项(#2)，因为我们仍需要从服务器端返回同样的数据结构(信息集包装在这个数据结构里)。

在服务器端，我们可以通过Sakila数据库视图sales_by_film_category查询图表数据：

```
SELECT * FROM sales_by_film_category
```

8.3.1 饼图

现在我们能够获取需要的信息了，接下来要实现图表。我们先来开发一个饼图：

```
Ext.define('Packt.view.reports.SalesFilmCategoryPie', {
    extend: 'Ext.chart.Chart',
    alias: 'widget.salesfilmcategorypie',

    animate: true,
    store: 'reports.SalesFilmCategory', // #1
    shadow: true,
    legend: {
        position: 'right' // #2
    },
    insetPadding: 60,
    theme: 'Base:gradients',
    series: [{
        type: 'pie', // #3
        field: 'total_sales', // #4
        showInLegend: true, // #5
        tips: {
              trackMouse: true, // #6
              width: 140,
              height: 28,
              renderer: function(storeItem, item) {
                this.setTitle(storeItem.get('category') + ': ' +
storeItem.get('total_sales'));
              }
        },
        highlight: {
          segment: {
            margin: 20
          }
```

```
            },
            label: {
                field: 'category', // #7
                display: 'rotate',
                contrast: true,
                font: '18px Arial'
            }
        }]
    });
```

我们先来分析上述代码中最重要的部分：首先，需要把刚才实现的存储器跟图表绑定在一起（#1）。接着，添加legend属性；在本项目中，我们希望图例显示在图表右边（#2）。

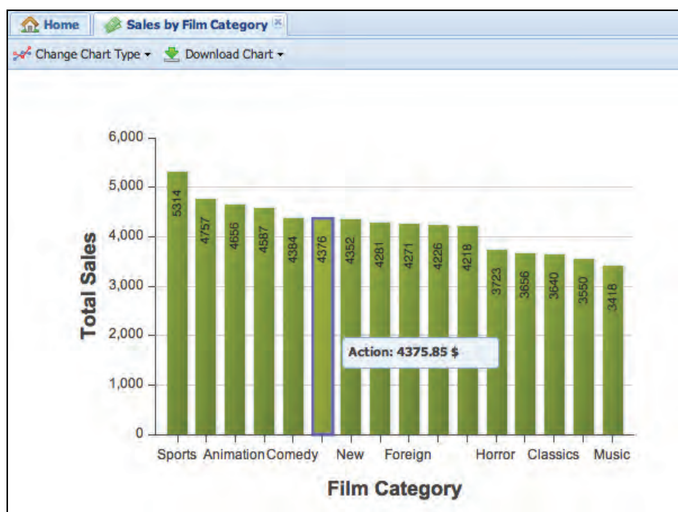
接下来是series属性配置项，用以定义图表类型（#3），本项目里是饼图。饼图需要一个字段用来计算（各部分的）总和，之后计算每部分的比例；我们只有两个字段（SalesFilmCategory存储器中），total_sales字段（#4）是一个数值型，因此将使用这个字段。showInLegend配置项用来控制是否在图例中添加元素（#5）。

我们在tips属性配置项里定义是否显示快速提示信息。在这里，我们希望Ext JS跟踪鼠标运动轨迹（#6），鼠标经过图表的各部分时，Ext JS会显示category字段名称及total_sales数值的提示信息。

最后，将label属性配置项设为category字段（#7），用来表示饼图的各个部分（在图表以及图例中）。

8.3.2　柱状图

由于可以更改图表类型，因此我们也可以实现一个如下图所示的柱状图：



接下来，我们就来实现这个柱状图：

```
Ext.define('Packt.view.reports.SalesFilmCategoryColumn', {
    extend: 'Ext.chart.Chart',
    alias: 'widget.salesfilmcategorycol',

    animate: true,
    store: 'reports.SalesFilmCategory', // #1
    shadow: true,
    insetPadding: 60,
    theme: 'Base:gradients',
    axes: [{
        type: 'Numeric', // #2
        position: 'left',
        fields: ['total_sales'], // #3
        label: {
            renderer: Ext.util.Format.numberRenderer('0,0')
        },
        title: 'Total Sales',
        grid: true,
        minimum: 0
    }, {
        type: 'Category', // #4
        position: 'bottom',
        fields: ['category'], // #5
        title: 'Film Category'
    }],
    series: [{
        type: 'column', // #6
        axis: 'left',
        highlight: true,
        tips: {
          trackMouse: true,
          width: 140,
          height: 28,
          renderer: function(storeItem, item) {
            this.setTitle(storeItem.get('category') + ': ' +
                storeItem.get('total_sales') + ' $');
          }
        },
        label: {
          display: 'insideEnd',
          'text-anchor': 'middle',
            field: 'total_sales',
            renderer: Ext.util.Format.numberRenderer('0'),
            orientation: 'vertical',
            color: '#333'
        },
        xField: 'category', // #7
        yField: 'total_sales' // #8
  }]
});
```

柱状图与饼图使用同一个存储器（#1）。由于柱状图是一个笛卡儿图，因此需要定义X轴和Y轴。我们在左边垂直放置数轴（#2），数轴用total_sales字段值表示（#3）。下一步是Category轴（#4），这是一个标签字段，代表图表的各列。Category轴用category字段值表示（#5）。

接下来我们定义series属性配置项。该属性定义待实现图表的类型，这里我们要实现的是一个柱状图（#6）。还需要分别定义*X*轴和*Y*轴字段，以便图表能够读取并应用每个轴的正确字段。*X*轴字段（水平方向的）用category字段表示（#7），*Y*轴字段（垂直方向的）用total_sales字段表示（#8）。有一点非常重要：xField（#7）匹配Category轴（#4），yField（#8）匹配数轴（垂直方向/位于左边：#2）。

条形图其实就是进行了微小变化的柱状图。我们需要转换坐标轴（Category轴在左边，数轴在底部）、xField属性（用total_sales替代category）以及yField属性（用category替代total_sales）。

条形图效果如下图所示：



8.3.3　图表面板

由于我们想要呈现一个面板并允许用户更改图表类型，因此，需要创建一个使用card（卡片式）布局的面板。要记住，card布局主要用于有好几个子项、但希望一次只显示一个子项的向导式情形。同时，当前显示项使用的是Fit布局。

现在来创建图表面板：

```
Ext.define('Packt.view.reports.SalesFilmCategory', {
    extend: 'Ext.panel.Panel',
    alias: 'widget.salesfilmcategory',

    layout: 'card',
    activeItem: 0,

    items: [
        {
            xtype: 'salesfilmcategorypie' // #1
        },
        {
            xtype: 'salesfilmcategorycol' // #2
        },
        {
            xtype: 'salesfilmcategorybar' // #3
        }
    ],

    dockedItems: [{
        xtype: 'toolbar',
        flex: 1,
        dock: 'top',
        items: [
            // items属性配置项声明 #4
        ]
    }]
});
```

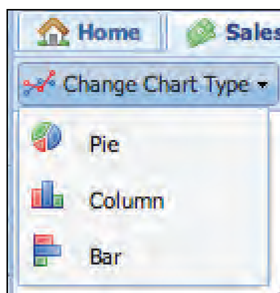
我们需要声明一个面板并声明已创建的每个图表为子项。因此，可以声明饼图（#1）、柱状图（#2）以及条形图（#3）作为Sales by Film Category（影片类别销售图面板）的子项。默认情况下，子项0（第一个子项：饼图）将成为图表面板渲染时呈现的默认选项。

接下来，我们将声明一个包含Menu（菜单）按钮的工具栏，以便用户可以选择图表类型以及下载保存的文件格式。因此，将以下代码放在上述代码里标识#4的位置上：

```
{
    text: 'Change Chart Type',
    iconCls: 'menu_reports',
    menu: {
        xtype: 'menu',
        itemId: 'changeType',
        items: [
            {
                xtype: 'menuitem',
                text: 'Pie',
                itemId: 'pie',
                iconCls: 'chart_pie'
            },
            {
```

```
                xtype: 'menuitem',
                text: 'Column',
                itemId: 'column',
                iconCls: 'chart_bar'
            },
            {
                xtype: 'menuitem',
                text: 'Bar',
                itemId: 'bar',
                iconCls: 'chart_column'
            }
        ]
    }
}
```

第一个声明的菜单按钮是Change Chart Type按钮，如下图所示：



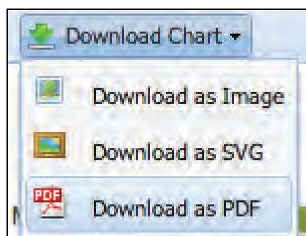
现在我们有了一个作为工具栏子项的Change Chart Type按钮（记住，这个按钮是工具栏子项的默认xtype），该按钮有一个包含三个菜单项的菜单，每个菜单项对应一种图表类型。

工具栏第二个子项是Download Chart按钮。与Change Chart Type按钮一样，Download Chart按钮也有一个包含3个菜单项的菜单，每个菜单项对应一种下载保存类型：

```
{
    text: 'Download Chart',
    iconCls: 'download',
    menu: {
        xtype: 'menu',
        itemId: 'download',
        items: [
            {
                xtype: 'menuitem',
                text: 'Download as Image',
                itemId: 'png',
                iconCls: 'image'
            },
            {
                xtype: 'menuitem',
                text: 'Download as SVG',
                itemId: 'svg',
```

```
                    iconCls: 'svg'
                },
                {
                    xtype: 'menuitem',
                    text: 'Download as PDF',
                    itemId: 'pdf',
                    iconCls: 'pdf'
                }
            ]
        }
    }
```

Download Chart按钮的效果如下图所示：



8.3.4 更改图表类型

由于用户可通过选择Menu按钮的菜单选项更改图表类型，因此，我们首先需要监听menuitem（菜单项）的点击事件：

```
"salesfilmcategory menu#changeType menuitem": {
    click: this.onChangeChart
}
```

menuitem点击事件与按钮点击事件类似；不同点在于我们要找的菜单项在itemId为changeType的菜单里。当用户点击一个菜单项时，Films控制器将执行以下方法：

```
onChangeChart: function(item, e, options) {
    var panel = item.up('salesfilmcategory'); // #1

    if (item.itemId == 'pie'){
        panel.getLayout().setActiveItem(0); // #2
    } else if (item.itemId == 'column'){
        panel.getLayout().setActiveItem(1); // #3
    } else if (item.itemId == 'bar'){
        panel.getLayout().setActiveItem(2); // #4
    }
}
```

首先，我们需要获取面板，这样就能改变活动子项（#1）。然后，比较用户点击菜单项的itemId，并根据用户选择设置活动子项（#2、#3和#4）。

8.3.5　图表导出成图片格式（PNG和SVG）

如同Change Chart Type菜单按钮，我们将在控制器上按同样的逻辑监听事件，不同之处在于我们监听的菜单项属于itemId属性设置为download的菜单：

```
"salesfilmcategory menu#download menuitem": {
    click: this.onChartDownload
}
```

下面的onChartDownload方法与Change Chart Type菜单项有相同的处理逻辑。但在这里，我们要做的是将图片保存为PNG或SVG文件。

```
onChartDownload: function(item, e, options) {
    var chartPanel = item.up('salesfilmcategory');
    var chart = chartPanel.getLayout().getActiveItem(); // #1

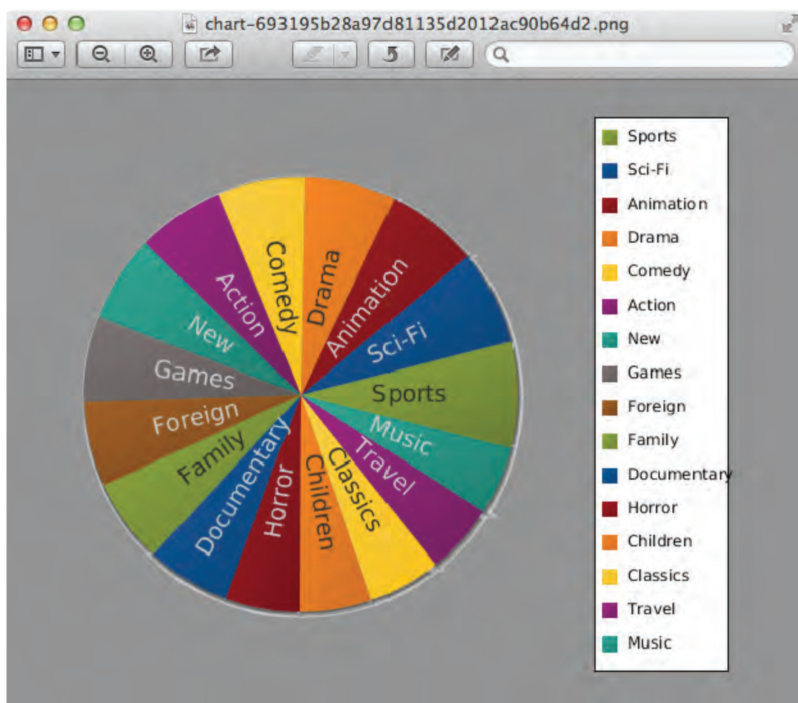
    if (item.itemId == 'png'){
        Ext.MessageBox.confirm('Confirm Download',
'Would you like to download the chart as Image?', function(choice){
            if(choice == 'yes'){
                chart.save({     // #2
                    type: 'image/png'
                });
            }
        });
    } else if (item.itemId == 'svg'){
        Ext.MessageBox.confirm('Confirm Download',
'Would you like to download the chart as SVG + XML?', function(choice){
            if(choice == 'yes'){       // #3
                chart.save({
                    type: 'image/svg+xml'
                });
            }
        });
    }
    // 下载保存成PDF文件（下一节实现）
}
```

chart类已有个名为save的方法了，我们可以用来将图表下载保存为图片格式。这是Ext JS提供的原生功能。

首先，我们需要获取图表引用，可以通过图表面板chartPanel的活动子项获得（#1）。

接下来，根据用户选择，先询问用户是否确定将图表下载保存为特定格式，如果确定，Ext JS将生成这个文件。当用户下载保存PNG（#2）或SVG（#3）文件时，我们只需调用chart引用的save方法，并传入用户选择的特定类型（PNG或SVG）即可。在这里，应用程序将发送一个请求到http://svg.sencha.io，然后开始下载。

以下截图是选择将图表保存为PNG格式文件时生成的图片：



8.3.6 图表导出成PDF格式

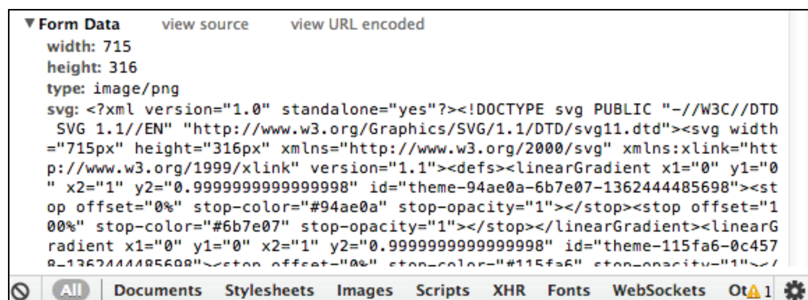
接下来完成本章最后一个功能的代码：

```
else if (item.itemId == 'pdf'){
    Ext.MessageBox.confirm('Confirm Download',
'Would you like to download the chart as PDF?', function(choice){
        if(choice == 'yes'){
            chart.save({    // #4
                type: 'image/png',
                url: 'php/pdf/exportChartPdf.php'
            });
        }
    });
}
```

在标识#4处可以看到，我们调用了chart类的save方法，同时还传入了一个URL。

如果我们进一步分析save方法，就会发现它调用了Ext.draw.engine.ImageExporter类的generate方法。ImageExporter类使用defaultUrl作为请求发送的目标URL，我们可以对其进行自定义。如果我们观察下请求，就会发现它发送了四个参数到服务器端：width、height、type和svg：

8



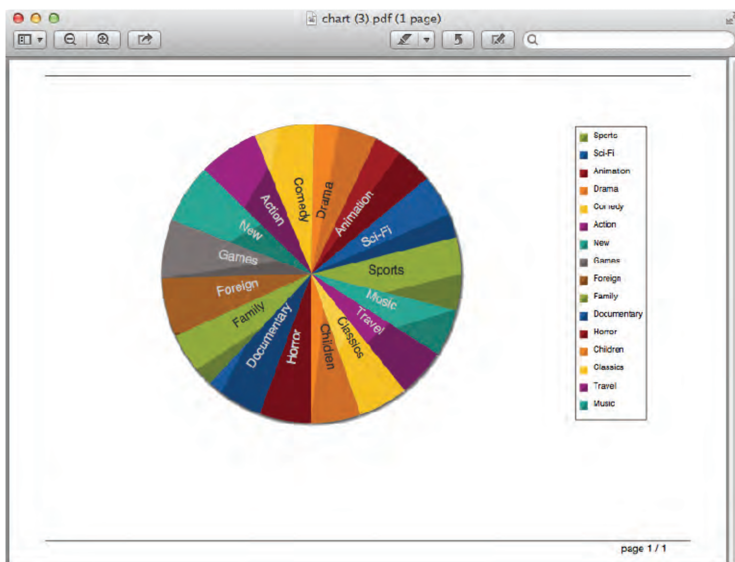
因此，我们需要在服务器端获取这些通过POST请求发送过来的信息项：

```
$width = $_POST['width'];
$height = $_POST['height'];
$type = $_POST['type'];
$svg = $_POST['svg'];
```

通过TCPDF类库，我们可以使用ImageSVG方法并传入这些参数：

```
$pdf->ImageSVG($file='@'.$svg, $x=10, $y=10, $w=$width, $h=$height, $link='',
$align='T', $palign='C', $border=0, $fitonpage=true);
```

奇迹发生了！生成的PDF文件如下图所示：



如果存在某些原因使应用程序部署的环境无法提供跨域（如http://svg.sencha.io）请求，你可以使用同样方式生成PNG、JPEG或SVG格式文件。Ext JS总是发送上述四个参数给服务器端；我们只需要根据实际来处理它们即可。

8.4　小结

本章我们学习了如何将网格面板的内容导出成PDF、Excel格式，以及如何生成打印页面。

我们学习了如何创建不同类型的图表，只使用一个组件（图表面板`SalesFilmCategory`组件）并改变其活动子项、借助Ext JS原生功能将图表导出为图片或SVG文件。我们同时还掌握了将图表导出为PDF文件的方法。

到目前为止，我们完成了与`Sakila`数据库相关的应用系统功能的开发。下一章我们将学习如何创建一个看起来跟Outlook很像的电子邮件客户端。

第 9 章 电子邮件客户端模块

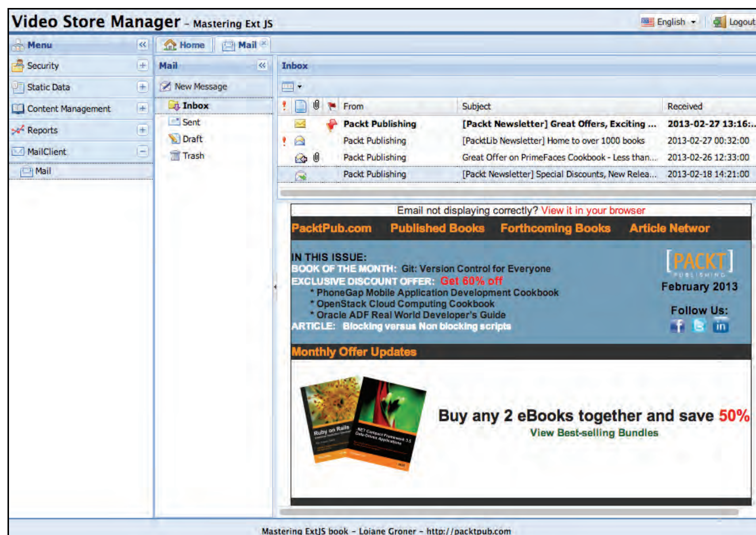
本章我们将实现应用程序的最后一个模块，参考Outlook（微软的一个非常流行的电子邮件客户端）的外观开发一个电子邮件客户端模块。

本章主要包括以下内容：

- ❑ 设计电子邮件客户端；
- ❑ 列出邮件；
- ❑ 创建收件箱菜单（树形面板菜单）；
- ❑ 将邮件拖放至新文件夹（网格与树形组件之间）；
- ❑ 优化网格面板。

9.1 创建收件箱：邮件列表

开始之前，我们先来看一看本章功能开发完成后的最终效果图：



这个电子邮件客户端由4个主要部分组成：Mail（邮件）菜单、Inbox（收件箱，邮件列表）、邮件预览主面板以及New Message（新建邮件）窗体。我们准备先实现收件箱功能，它是一个增强的网格面板。按照既定的开发流程，我们将首先创建模型，然后是存储器，再然后是视图，最后创建控制器并监听动作触发的事件。

9.1.1 邮件信息模型

首先，我们需要创建一个新模型Packt.model.mail.MailMessage，用以表示准备显示在网格面板上的邮件信息：

```
Ext.define('Packt.model.mail.MailMessage', {
    extend: 'Ext.data.Model',
    fields: [
        { name: 'importance' },
        { name: 'icon'  },
        { name: 'attachment'  },
        { name: 'from' },
        { name: 'subject' },
        { name: 'received' },
        { name: 'flag' },
        { name: 'folder' },
        { name: 'content' },
        { name: 'id' }
    ]
});
```

importance字段显示红色惊叹号标记，表示邮件以高优先级发送；icon字段是邮件信息的图标（已读、未读、转发、回复以及回复全部）；attachment字段表示邮件有附件；flag字段表示旗标状态（在Outlook里用作后续跟踪标志）；folder字段用来存放邮件（收件箱、草稿箱以及垃圾箱，诸如此类）。

这基本上就是一个简单的模型，跟我们前面几章实现的模型没什么区别。

9.1.2 邮件信息存储器

接下来，我们实现一个加载邮件信息的存储器。创建存储器类Packt.store.mail.MailMessages：

```
Ext.define('Packt.store.mail.MailMessages', {
    extend: 'Ext.data.Store',

    requires: [
        'Packt.model.mail.MailMessage',
        'Packt.proxy.Sakila'
    ],
```

```
        model: 'Packt.model.mail.MailMessage',
        autoLoad: true,

        proxy: {
            type: 'sakila',
            api: {
                read: 'php/mail/listInbox.php',
                update: 'php/mail/update.php'
            }
        }
    });
```

我们将从服务器端加载（read）邮件信息（出于示例的目的，不打算集成像Gmail、Hotmail/Outlook、Yahoo以及其他的邮件服务）。

如果改变了邮件文件夹，我们也将相应地修改（update）邮件信息。后续我们还会实现拖放功能。

在本例中，我们将从服务器端取回所有的模型信息。如果用户有大量邮件，存储器就需要一段时间来加载信息。就目前掌握的知识，我们可以采取两种不同的处理方式：第一种方式是实现一个分页工具栏；第二种方式是按需加载content（内容）字段，即只有当用户点击收件箱中的一个邮件时才读取其内容。

9.1.3　邮件列表视图

最后来实现视图，这是一个网格面板，我们将其命名为Packt.view.mail.MailList：

```
Ext.define('Packt.view.mail.MailList', {
    extend: 'Ext.grid.Panel',
    alias: 'widget.maillist',

    title: 'Inbox',
    store: 'mail.MailMessages',

    viewConfig: {
        getRowClass: function(record, rowIndex, rowParams, store){
            if (record.get('icon') == 'unread'){ // #1
                return "boldFont";
            }
        }
    },
    columns: [
        // 这里声明各列
    ]
});
```

上述代码中有一段需要我们注意：viewConfig属性配置项中的getRowClass函数。在这个函数里，我们可以返回一个CSS样式，这个样式将应用于网格面板上一条记录的所有单元格。在本例中，如果邮件是unread（未读）状态（#1），我们考虑通过应用boldFont样式使此行的字

体为粗体。如果已读，那么就什么也不用做了。

我们需要在app.css文件里声明这个样式：

```
.boldFont .x-grid-cell {
  font-weight:bold; !important;
}
```

接下来声明网格面板的各列。我们逐个实现它们，这样就可以详细掌握细节。首先，声明importance列，如果邮件以高优先级发送，那么它将显示一个红色的惊叹号标记。

```
{
    xtype: 'gridcolumn',
    cls: 'importance', // #2
    width: 18,
    dataIndex: 'importance',
    menuDisabled: true,
    text: 'Importance',
    renderer: function(value, metaData, record ){
        if (value == 1){
            metaData.css = 'importance'; // #3
        }
        return '';
    }
}
```

这里有两个需要重点关注的地方：观察一下前面的电子邮件客户端的完整截图，我们会注意到网格头部不是文字而是图标。怎样在网格头部显示一个图标呢？很简单，只需在columns里声明cls属性，应用CSS样式到网格头部即可（#2）。

importance CSS代码如下：

```
.importance {
  background:transparent url('../icons/mail/priority_high.gif') no-repeat 3px
3px !important;
  text-indent:-250px;
}
```

如果邮件按高优先级发送，我们希望在网格的单元格里显示一个图标而非文字。要想实现这个目的，也需在renderer函数里应用importance CSS样式到网格单元格上（#3）。然后返回一个空字符串，因为我们并不想显示文字。

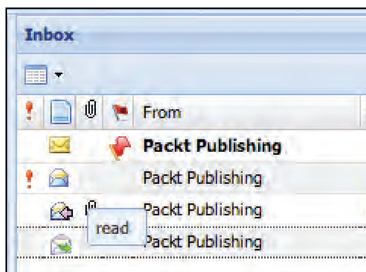
接下来，我们声明显示图标消息的icon列，换言之，它告诉用户邮件是哪种类型：已读、未读、回复、转发或是回复全部：

```
{
    xtype: 'gridcolumn',
    cls: 'icon-msg', // #4
    width: 21,
    dataIndex: 'icon',
```

```
        menuDisabled: true,
        text: 'Icon',
        renderer: function(value, metaData, record ){
            metaData.css = value; // #5
            metaData.tdAttr = 'data-qtip="' + value + '"'; // #6
            return '';
        }
    }
```

我们将再次在网格头部显示一个图标，因此需要声明cls属性（#4）。同时还要在网格单元格里显示一个图标，但这次，该列的CSS将是从数据库取回的列值本身（#5）。

当用户的鼠标悬停在网格单元格上时，需要显示一个快速提示信息。我们同样可以通过设置metadata的tdAttr属性值来实现这一功能（#6）。本例中，快速提示信息将显示数据库获取的列值，结果如下图所示：



attachment列的代码与前面几列的很相似：

```
    {
        xtype: 'gridcolumn',
        cls: 'attach', // #7
        width: 18,
        dataIndex: 'attachment',
        menuDisabled: true,
        text: 'Attachment',
        renderer: function(value, metaData, record ){
            if (value == 1){
                metaData.css = 'attach'; // #8
            }
            return '';
        }
    }
```

我们打算在网格头部（#7）、网格单元格中（#8）显示一个图标。

flagged列也一样：

```
    {
        xtype: 'gridcolumn',
        cls: 'flagged', // #9
```

```
        width: 20,
        dataIndex: 'flag',
        menuDisabled: true,
        text: 'Flag',
        renderer: function(value, metaData, record ){
            if (value == 1){
                metaData.css = 'flag-e-mail'; // #10
            }
            return '';
        }
    }
```

我们打算在网格头部（#9）、网格单元格中（#10）显示一个图标。

下一步声明剩余的几列，在网格头部（通常情况下）、网格单元格内容里显示文字：

```
    {
        xtype: 'gridcolumn',
        dataIndex: 'from',
        menuDisabled: true,
        width: 150,
        text: 'From'
    },
    {
        xtype: 'gridcolumn',
        dataIndex: 'subject',
        menuDisabled: true,
        flex: 1,
        text: 'Subject'
    },
    {
        xtype: 'gridcolumn',
        dataIndex: 'received',
        menuDisabled: true,
        width: 130,
        text: 'Received'
    }
```


当前代码的输出结果如下图所示，这是一个对网格头部、网格单元格进行了个性化设置，同时在网格记录上应用了自定义CSS样式的网格面板。所有这些功能都是靠原生技术实现的，不需要第三方插件：

Inbox

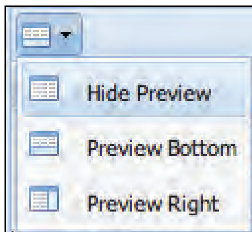
				From	Subject	Received
				Packt Publishing	**[Packt Newsletter] Great Offers, Exciting Ne...**	**2013-02-27 13:16:...**
				Packt Publishing	[PacktLib Newsletter] Home to over 1000 books	2013-02-27 00:32:00
				Packt Publishing	Great Offer on PrimeFaces Cookbook - Less than 24 ...	2013-02-26 12:33:00
				Packt Publishing	[Packt Newsletter] Special Discounts, New Releases...	2013-02-18 14:21:00

9.1.4　邮件预览面板

还剩下一个细节需要处理，那就是可以在前面的截图上看到的菜单按钮：

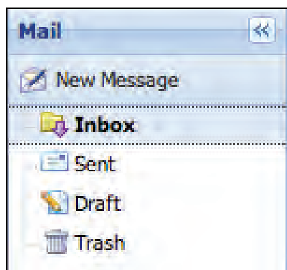
```
dockedItems: [
    {
        xtype: 'toolbar',
        dock: 'top',
        items: [
            {
                xtype: 'button',
                iconCls: 'preview-hide',
                menu: {
                    xtype: 'menu',
                    itemId: 'preview',
                    width: 120,
                    items: [
                        {
                            xtype: 'menuitem',
                            itemId: 'hide',
                            iconCls: 'preview-hide',
                            text: 'Hide Preview'
                        },
                        {
                            xtype: 'menuitem',
                            itemId: 'bottom',
                            iconCls: 'preview-bottom',
                            text: 'Preview Bottom'
                        },
                        {
                            xtype: 'menuitem',
                            itemId: 'right',
                            iconCls: 'preview-right',
                            text: 'Preview Right'
                        }
                    ]
                }
            }
        ]
    }
]
```

这个菜单允许用户改变预览标签表单的位置（右边或底部），甚至隐藏预览面板（Hide Preview），如下图所示：



9.2　邮件菜单（树形菜单）

下一步，我们将创建列出收件箱（Inbox）里的文件夹的菜单，如下图所示：



9.2.1　树形邮件菜单存储器

按照惯例，我们从创建模型和存储器入手。由于使用NodeInterface类表示每个node（树节点缺省类），因此我们不再需要进行任何自定义——不必声明模型[①]，可以直接实现树形菜单的存储器：

```
Ext.define('Packt.store.mail.MailMenu', {
    extend: 'Ext.data.TreeStore',

    clearOnLoad: true,

    proxy: {
        type: 'ajax',
        url: 'php/mail/mailMenu.php',
    }
});
```

我们将从服务器端加载JSON格式数据。相关的PHP代码很简单，只是返回所需JSON数据（已被硬编码成一个字符串了）：

```
<?php
echo '[{
    "text": "Inbox",
    "iconCls": "folder-inbox",
    "leaf": true
},{
    "text": "Sent",
    "iconCls": "folder-sent",
    "leaf": true
},{
    "text": "Draft",
```

① 树节点是用NodeInterface包装好的一个模型实例。——译者注

9

```
        "iconCls": "folder-drafts",
        "leaf": true
    },{
        "text": "Trash",
        "iconCls": "folder-trash",
        "leaf": true
    }]';
    ?>
```

9.2.2　创建邮件菜单视图

接下来创建表示邮件菜单的树形面板，这是一个很简单的树形面板：

```
Ext.define('Packt.view.mail.MailMenu', {
    extend: 'Ext.tree.Panel',
    alias: 'widget.mailMenu',

    cls: 'selected-node', // #1
    autoScroll: true,
    store: 'mail.MailMenu',
    rootVisible: false,
    split: true,
    width: 150,
    collapsible: true,
    title: 'Mail',

    dockedItems: [
        {
            xtype: 'toolbar', // #2
            dock: 'top',
            items: [
                {
                    xtype: 'button',
                    iconCls: 'new-mail',
                    text: 'New Message',
                    itemId: 'newMail'
                }
            ]
        }
    ]
});
```

这里有两点需要注意,第一点是当用户从树形面板上选择一个节点时,我们希望它加粗显示，因此，需要应用一个CSS（#1）样式来实现：

```
.selected-node .x-grid-row-selected .x-grid-cell {
    font-weight: bold;
}
```

第二点是实现了一个带有New Message（撰写新邮件）按钮的工具栏（#2），用户可以撰写及发送新邮件。

9.3　邮件容器：组织电子邮件客户端

目前，我们已经实现了两个主要的电子邮件客户端模块。现在需要组织它们，以使界面看起来如本章开头的截图所示。要达成此目标，我们需要添加一个使用边界布局的容器。



为什么要使用容器而非面板呢？容器是个比面板更轻量级的组件。我们不需要使用诸如面板头部和DockedItem此类的面板功能，只想把组件包装起来并用特定布局组织它们而已。因此，在本例中，容器是更好的选择。

来看看MailContainer组件的代码：

```
Ext.define('Packt.view.mail.MailContainer', {
    extend: 'Ext.container.Container',
    alias: 'widget.mailcontainer',

    requires: [ // #1
        'Packt.view.mail.MailList',
        'Packt.view.mail.MailPreview',
        'Packt.view.mail.MailMenu'
    ],

    layout: {
        type: 'border' // #2
    },

    initComponent: function() {
        var me = this;
        var mailPreview = { // #3
            xtype: 'mailpreview',
            autoScroll: true
        };

        me.items = [
            {
                xtype: 'container', // #4
                region: 'center',
                itemId: 'mailpanel',
                layout: {
                    type: 'border'
                },
                items: [
                    {
                        xtype: 'maillist',
                        collapsible: false, // #5
                        region: 'center'
                    },
                    {
```

9

```
                    xtype: 'container',
                    itemId: 'previewSouth', // #6
                    height: 300,
                    hidden: true,
                    collapsible: false,
                    region: 'south',
                    split: true,
                    layout: 'fit',
                    items: [mailPreview]
                },
                {
                    xtype: 'container',
                    width: 400,
                    itemId: 'previewEast', // #7
                    hidden: true,
                    collapsible: false,
                    region: 'east',
                    split: true,
                    layout: 'fit',
                    items: [mailPreview]
                }
            ]
        },
        {
            xtype: 'mailMenu', // #8
            region: 'west',

        }
    ];

    me.callParent(arguments);
  }
});
```

要牢记，我们自己创建（非原生既有）、准备用其xtype来实例化的类一定要在requires中声明（#1）。

我们将使用边界布局（#2）。目的是把邮件菜单放在容器左侧，邮件列表和预览窗体放在中间。

由于我们打算声明一个邮件预览容器，可根据用户选择显示在邮件列表的底部或右侧，并避免代码重复，因此，我们只声明它一次（#3），并在#6和#7处重用它。

接下来是邮件面板，用来组织邮件列表和邮件预览容器。邮件面板（#4）也使用边界布局，其子项分别位于中央位置（中央区域是强制性的）、底部和右侧（邮件预览容器）。

邮件列表子项（#5）位于邮件面板中央。邮件预览容器位于邮件面板底部（#6）或右侧（#7）。我们将使用在邮件列表中声明的菜单按钮来控制邮件预览容器的隐藏与显示。最终，我们将获得创建完成的邮件菜单（#8）。

现在，唯一还未实现的是邮件预览组件，它也是个容器，并使用fit布局（我们希望邮件内

容占满容器的剩余空间）：

```
Ext.define('Packt.view.mail.MailPreview', {
    extend: 'Ext.container.Container',
    alias: 'widget.mailpreview',

    layout: 'fit'
});
```

9.4 控制器

现在，我们需要的东西都准备好了，接下来实现允许用户选择邮件预览面板位置的功能。用户可以选择将预览面板放在右侧或底部，甚至隐藏它。

因此，第一步要做的是为电子邮件客户端模块创建一个新的控制器：

```
Ext.define('Packt.controller.mail.Mail', {
    extend: 'Ext.app.Controller',

    views: [ // #1
        'mail.MailContainer',
        'mail.MailList',
        'mail.MailPreview'
    ],

    stores: [// #2
        'mail.MailMessages',
        'mail.MailMenu'
    ],

    refs: [// #3
        {
            ref: 'south', // #4
            selector: 'mailcontainer container#previewSouth'
        },
        {
            ref: 'east', // #5
            selector: 'mailcontainer container#previewEast'
        }
    ]
});
```

首先，要记得声明我们为模块创建的视图（#1）。之后是声明存储器（#2）。最后，需要声明某些引用（#3）：声明位于邮件容器底部（#4）和右侧（#5）的邮件预览容器的引用。

要记住，引用始终通过选择器创建组件快捷方式。我们可以声明引用并使用诸如`this.getSouth`的简单调用，避免每次都通过`Ext.ComponentQuery.query('mailcontainer container#previewSouth)[0]`来获取引用。同时，如果将来需要改变选择器，这么做便于我们进行维护管理，因为有了这个引用，我们只需在一个地方改变选择器即可。

电子邮件预览

接下来，我们需要监听menuitem的点击事件：

```
"menu#preview menuitem": {
    click: this.onMenuitemClick
}
```

我们使用前几章监听menuitem点击事件时采取的方法。为了避免一个个地监听每个菜单项，我们可以监听特定菜单的所有菜单项，之后，在方法里可以通过itemId找到哪个菜单项被点击了：

```
onMenuitemClick: function(item, e, options) {

    var button = item.up('button'); // #1
    var east = this.getEast();      // #2
    var south = this.getSouth();    // #3

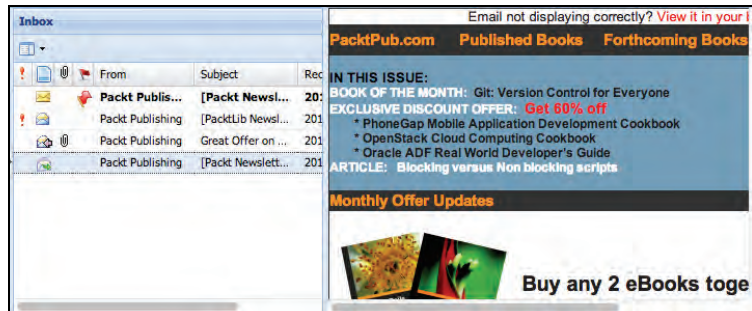
    switch (item.itemId) {
        case 'bottom': // #4
            east.hide();
            south.show();
            button.setIconCls('preview-bottom');
            break;
        case 'right': // #5
            south.hide();
            east.show();
            button.setIconCls('preview-right');
            break;
        default:      // #6
            south.hide();
            east.hide();
            button.setIconCls('preview-hide');
            break;
    }
}
```

上述代码的实现思路是更改点击菜单按钮的图标，并根据用户选择，呈现或隐藏底部和右侧的邮件预览容器。

因此，第一步是获取菜单按钮的引用（#1），以便后续可以更改它的iconCls属性。接下来，需要获取底部（#3）和右侧（#2）的邮件预览容器的引用。

如果用户选择在底部呈现预览容器（#4），就得隐藏右侧容器并显示底部容器。如果用户选择在右侧呈现预览容器（#5），就显示右侧容器并隐藏底部容器。如果用户选择隐藏预览容器（#6），底部容器和右侧容器就都被隐藏。

比如，如果用户选择在右侧呈现预览容器，其效果如下图所示：



9.5　组织电子邮件：拖放

在Outlook中，可以选中一封邮件并将其拖放到另一文件夹里。我们也将在我们的电子邮件客户端模块里实现这个功能。但有一个细节需要注意：首先，我们需要在网格面板（邮件列表）和树形面板（邮件菜单）之间实现拖放；其次，我们并不想把邮件从网格面板移动到树形面板中。我们只想简单地把邮件归（注意：不是放）至树形面板的一个节点，并非真要把邮件当成新节点添加到树形面板中（而这意味着要把这封邮件添加到树形面板的存储器中）。因此，当实现该功能时还请记住上述这一点。这个例子非常好，因为它说明我们可以自定义拖放功能的一些动作。

首先，我们需要给网格面板和树形面板添加拖放功能。

在Packt.view.mail.MailList类的viewConfig方法中，我们需要添加以下代码（在getRowClass函数之前或之后，根据个人意愿）：

```
plugins: {
    ptype: 'gridviewdragdrop',
    ddGroup: 'mailDD'
}
```

我们添加了一个拖放插件（Ext JS的原生功能），并为ddGroup属性设置名称值，ddGroup属性配置项表示插件作用的拖放区域[①]。如果我们在应用程序里实现了另一个具有不同名称的拖放（区域），无法实现从本网格面板（拖放区域）到另一拖放区域的拖放操作。

接下来，在Packt.view.mail.MailMenu类里添加以下代码：

```
viewConfig: {
    plugins: {
        ptype: 'treeviewdragdrop',
        ddGroup: 'mailDD',
        enableDrag: false
    }
}
```

① 即拖放操作在此区域范围内有效。——译者注

我们还实现了一个针对树形面板的拖放插件。这里我们并不启用拖曳动作，请注意本例的ddGroup属性配置项的名称值跟网格面板里的ddGroup属性是一样的，这就意味着网格面板可跟树形面板进行拖放操作。

到目前，拖放还未实现。我们回到控制器来实现拖放程序逻辑。

首先，需要监听树形面板的treeview触发的beforedrop事件。

由于表单组件包含了表单基本类、树形面板包含了树形视图（treeview）类、网格面板包含了网格视图（gridview）类。因此，视图是实际负责内容的组件，网格或树是包含了视图并提供其他功能的组件。

```
"mailMenu treeview": {
    beforedrop: this.onBeforeDrop
}
```

下面的代码片段实现了onBeforeDrop方法：

```
onBeforeDrop: function(node, data, overModel, dropPosition, dropHandler, options) {
    Ext.each(data.records, function(rec){ // #1
        rec.set('folder',overModel.get('text')); // #2
    });
    dropHandler.cancelDrop();// #3

    var grid= Ext.ComponentQuery.query('maillist')[0];
    var store = grid.getStore();
    store.sync();       // #4
}
```

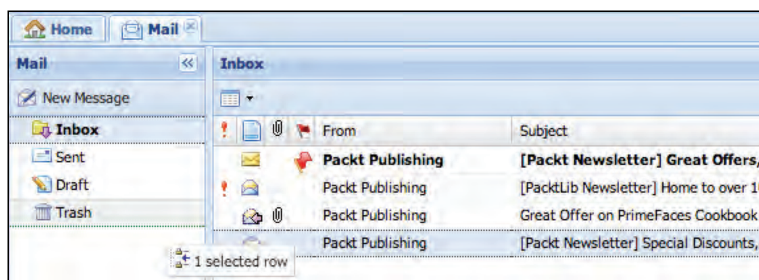
在data参数中，可以发现有个records属性，该属性包含了拖曳到树节点的记录。因此我们需要循环处理所有的记录（#1）。

这个"处理"是指改变模型中的folder字段。记住，我们并不需要从网格面板上移除记录，只需把电子邮件从一个文件夹里归到另一个文件夹里即可（#2，改变邮件所属文件夹）。要获取记录拖放至的节点的名称，我们可以通过overModel参数获取（overModel.get）需要的字段。

记住，不需要在树形面板中追加新节点，我们只是想更改被拖曳模型的folder字段（把邮件记录归至新的folder中）。因此，可以取消被触发的放入（drop）事件（#3）。要达成此目的，我们可以使用dropHandler参数，该参数包含了完成或取消数据传输操作的方法（不管是从源视图的存储器移动还是复制模型实例到目标视图的存储器）。

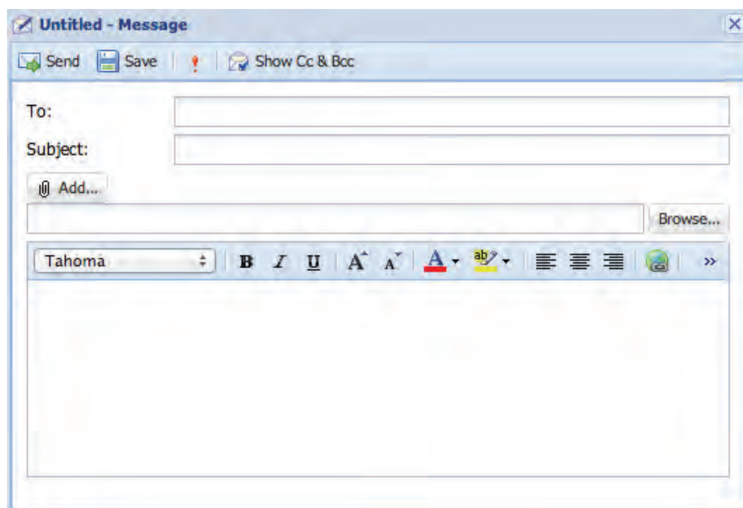
之后，调用网格面板存储器的sync方法，在数据库中保存邮件所属新文件夹的名称（#4）。

从网格面板拖曳邮件到树形面板的操作如下图所示：



9.6　创建新邮件

现在我们来实现最后部分的窗体功能，用户可以在这里撰写要发送的新邮件。编码之前，来看一下本节代码实现的效果图：



首先，我们需要创建包含撰写新邮件表单的窗体：

```
Ext.define('Packt.view.mail.NewMail', {
    extend: 'Ext.window.Window',
    alias: 'widget.newmail',

    height: 410,
    width: 670,
    autoShow: true,
    layout: {
        type: 'fit'
    },
    title: 'Untitled - Message',
    iconCls: 'new-mail'
});
```

截至当前，一切都很顺利。我们接下来声明dockedItems，其中将声明一个带有Send（发送）、Save（保存）、Importance（高优先级）以及Show Cc & Bcc（显示抄送和密送）等按钮的工具栏。

```
dockedItems: [
    {
        xtype: 'toolbar',
        dock: 'top',
        items: [
            {
                xtype: 'button',
                text: 'Send',
                iconCls: 'send-mail',
                itemId: 'send'
            },
            {
                xtype: 'button',
                text: 'Save',
                iconCls: 'save'
            },
            {
                xtype: 'tbseparator'
            },
            {
                xtype: 'button',
                iconCls: 'importance'
            },
            {
                xtype: 'tbseparator'
            },
            {
                xtype: 'button',
                text: 'Show Cc & Bcc',
                iconCls: 'bcc',
                itemId: 'bcc'
            }
        ]
    }
]
```

注意，`importance`按钮没有`text`属性配置项。`text`属性是可选的。由于我们只想显示图标，因此只需要设置`iconCls`属性配置项就够了。

现在来实现表单及其`items`子项：

```
items: [
    {
        xtype: 'form',
        frame: false,
        bodyPadding: 10,
        autoScroll: true,
        defaults: {
            anchor: '100%',
```

```
            xtype: 'textfield'
        }
        items: [
            {
                fieldLabel: 'To',
                name: 'to'
            },
            {
                fieldLabel: 'Cc', // #1
                hidden: true,
                name: 'cc'
            },
            {
                fieldLabel: 'Bcc', // #2
                hidden: true,
                name: 'bcc'
            },
            {
                fieldLabel: 'Subject',
                name: 'subject'
            },
            {
                xtype: 'button', // #3
                text: 'Add...',
                iconCls: 'attach',
                itemId: 'attach'
            },
            {
                xtype: 'filefield',
                name: 'file'
            },
            {
                xtype: 'htmleditor',
                height: 168,
                style: 'background-color: white;',
                name: 'content'
            }
        ]
    }
]
```


`Cc`（#1）和`Bcc`（#2）字段将被隐藏，因为我们希望当用户点击Show Cc & Bcc按钮时才呈现它们。

我们还实现了一个`Add`（添加）按钮（#3），以便用户可以在电子邮件中添加一个新附件。

9.6.1 动态呈现Cc和Bcc字段

由于我们创建了一个按钮来呈现`Cc`和`Bcc`字段，因此需要在控制器中监听按钮触发的点击事件。用户点击该按钮时，将执行以下方法：

```
onShowBcc: function(button, e, options){
    Ext.ComponentQuery.query('textfield[name=cc]')[0].show();
    Ext.ComponentQuery.query('textfield[name=bcc]')[0].show();
}
```

在这里，我们唯一要做的就是获取这两个字段的引用，并调用show方法。

输出效果图如下所示：

9.6.2　动态添加文件上传字段

我们已经实现了Add按钮，让用户可以在电子邮件中添加新附件。

因此，当用户点击Add按钮时，我们想要做的就是在已有的文件上传（fileUpload）字段后面，添加一个新的文件上传类。我们在控制器里实现该方法：

```
onNewAttach: function(button, e, options){
    var form = button.up('window').down('form');

    var fileUpload = {
        xtype: 'filefield',
        name: 'file' + this.attachPosition // #1
    };

    form.insert(this.attachPosition++, fileUpload); // #2
}
```

以上代码中有两点需要注意。第一点是新建文件上传字段的名称。已生成的文件上传字段已经有了名称file，因此就不能重名了。我们回顾一下，name（名称）属性是发送至服务器端的参数的名称。同时，用户可以按需添加多个新附件。因此，得想一个办法为每个文件上传组件创建唯一的名称（#1）。第二点需要关注的是，我们需要在控制器里创建一个新属性，同时把新建文件上传字段的插入位置赋值给该属性（#2）。新建文件上传字段将被插到已生成的初始文件上传字段之后、HTML编辑器（htmleditor）字段之前，因此，如果算一算，新建文件上传字段的插入位置应该是6：

```
attachPosition: 6
```

另外，每当用户新加一个文件上传字段，我们都需要递增插入位置值（this.attachPosition++）。

当用户关闭撰写新邮件窗体并再次打开它时，我们需要重将attachPosition值设为6：

```
onNewMessage: function(button, e, options){
    Ext.create('Packt.view.mail.NewMail');
    this.attachPosition = 6;
}
```

如果执行现有代码并点击Add按钮，新建文件上传字段将被插入到撰写新邮件表单中，并且用户可以添加新附件，效果图如下所示（添加了三个新字段）：

Add...
C:\fakepath\DSCN2447.JPG Browse...
C:\fakepath\DSCN2445.JPG Browse...
C:\fakepath\DSCN2316.JPG Browse...
C:\fakepath\DSCN2264.JPG Browse...

9.7 小结

本章实现了一个与Outlook电子邮件客户端软件非常相似的电子邮件客户端模块。这说明我们可以用Ext JS（其非常灵活）来干很多漂亮的事儿；我们只需要关注一些细节就可以了。

此外，我们还在网格面板和树形面板之间实现了拖放能力，并进行了一些自定义开发。

到本章为止，我们就完成了应用程序的所有功能。下一章，我们将学习如何自定义应用程序的外观（创建一个新的主题），并阐述如何优化程序并做好产品化准备。

第 10 章 产品化准备

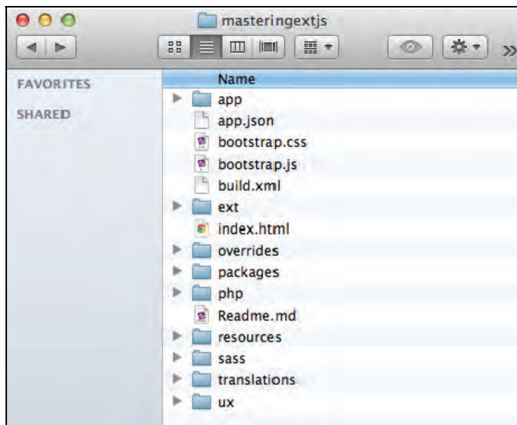
截至当前，应用程序的所有功能都完成了。现在我们来创建个漂亮的主题，突出应用程序的个性化风格，并为产品发布做些准备工作。毕竟我们所有的工作都是在开发环境下进行的，当产品要发布上线时，不能简单地部署所有文件就完事了，还需要提前做一些准备工作。因此，本章主要包括以下内容：

- ❑ 创建自定义主题；
- ❑ 为产品发布打包应用；
- ❑ 使用Sencha Desktop Packager（桌面打包工具）。

10.1　开始之前

本章使用的主要工具是Sencha Cmd（Sencha命令行工具）。通过此工具我们可以创建自定义主题，并进行产品构造。由于我们使用Ext JS 4.2，因此应该选择Sencha Cmd 3.1.1，它兼容Ext JS 4.2。我们始终要确保使用的Sencha Cmd版本与Ext JS版本兼容。

目前为止，本书已经完成了如下开发内容：



我们开发的所有代码都在app目录、index.html文件、php目录、resources目录（CSS文件及自定义图片图标文件）、translations目录以及ux目录（项目中用到的第三方插件）里。通过Sencha Cmd创建的其他目录和文件，我们都已在第一章了解到。

10.2　自定义主题

本章要做的第一件事就是自定义一个应用主题，我们需要通过Sencha Cmd和操作系统的终端应用程序来完成它。

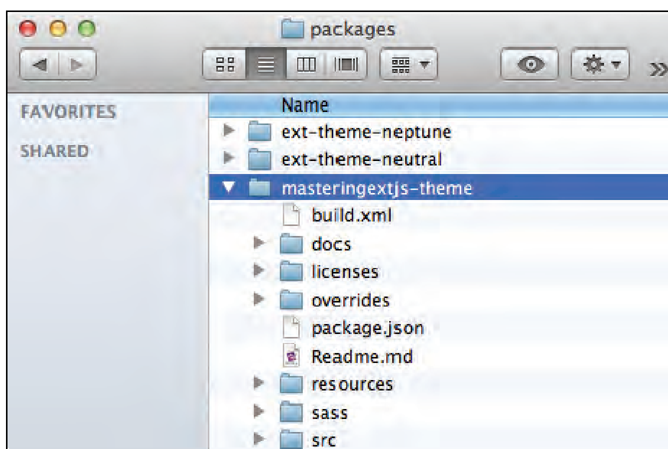
Ext JS 4.2提供了一个与之前Ext JS 4版本不同的新的自定义主题方式，根据创建全新主题所需，Sencha Cmd现在已具备生成相应完整文件结构的能力。我们接下来介绍这个新的自定义主题方式。

我们一步步地创建这个新主题。首先，打开终端应用程序，将目录切换到项目根目录，然后执行以下命令：

```
sencha generate theme masteringextjs-theme
```

```
masteringextjs — bash — 80×24
loiane:~ loiane$ cd /Applications/XAMPP/xamppfiles/htdocs/masteringextjs
loiane:masteringextjs loiane$ sencha generate theme masteringextjs-theme
Sencha Cmd v3.1.0.256
[INF]
[INF] init-plugin:
[INF]
[INF] init-plugin:
[INF] Invoking plugin (/Applications/XAMPP/xamppfiles/htdocs/masteringextjs/.sencha/app/plugin.xml) - supported targets: -before-generate-theme
[INF]
[INF] -before-generate-theme:
[INF] Invoking plugin (/Applications/XAMPP/xamppfiles/htdocs/masteringextjs/.sencha/app/plugin.xml) - supported targets: generate-theme
[INF]
[INF] cmd-root-plugin.init-properties:
[INF]
[INF] init-properties:
[INF]
[INF] init sencha command:
[INF]
[INF] init:
[INF]
[INF] -before-generate-theme:
[INF]
```

我们的主题名称为`masteringextjs-theme`。这条命令将在packages文件夹里创建一个新的目录，目录名与主题名称相同（`masteringextjs-theme`），如下图所示：



package.json文件包含了一些Sencha Cmd会用到的主题配置项，如主题名称、版本以及依赖关系等。

sass目录里包含了主题所需的所有Sass文件。在这个目录里，我们可以看到三个主要的目录。

- **var**　包含Sass变量。
- **src**　包含Sass规则和混入类（mixins）。这些规则和混入类使用sass/var目录文件里声明的变量。
- **etc**　包含额外的实用函数和混入类。

我们创建的所有文件必须匹配样式设计对应组件的类路径。比如想设计按钮组件，就需要在sass/var/button/Button.scss文件里创建其样式；如果想设计面板组件，就应该在sass/var/panel.scss文件里创建其样式。

resources目录包含主题要用到的图片及其他静态资源。

overrides目录包含了所有用于覆写组件的JavaScript代码，当对这些组件进行自定义主题时可能要用到这些代码。

花些时间解释接下来一些目录的内容，可以在packages文件夹里找到我们更熟悉的组织Sass文件的方式：ext-theme-classic、ext-theme-gray以及ext-theme-neptune。

默认情形下，我们创建的主题将使用ext-theme-classic文件（经典的Ext JS蓝色主题）作为基准主题。我们可以尝试一下Ext JS 4.2新的Neptune主题（代号：海神）。要改变基准主题，请打开package.json文件并定位到extend属性。将属性值由ext-theme-classic更改为ext-theme-neptune。package.json文件内容如下：

```
{
    "name": "masteringextjs-theme",
```

```
    "type": "theme",
    "creator": "anonymous",
    "version": "1.0.0",
    "compatVersion": "1.0.0",
    "local": true,
    "requires": [],
    "extend": "ext-theme-neptune"
}
```

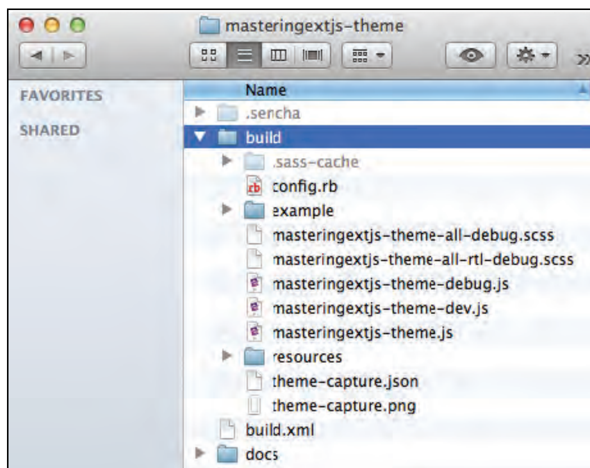
创建主题结构并且改变基准主题之后，我们来生成主题。再次通过终端应用程序和Sencha Cmd来完成此步骤。切换目录至packages/masteringextjs-theme，并执行以下命令：

```
sencha package build
```

结果如下图所示：

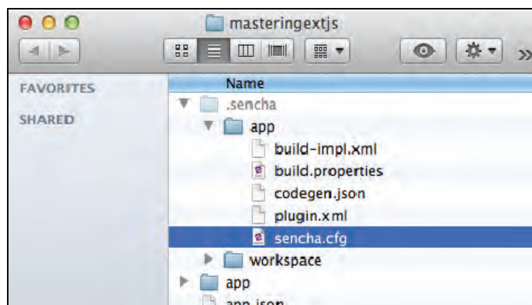
```
masteringextjs-theme — bash — 80×24
loiane:masteringextjs loiane$ cd packages/masteringextjs-theme
loiane:masteringextjs-theme loiane$ sencha package build
Sencha Cmd v3.1.0.256
[INF]
[INF] init-plugin:
[INF]
[INF] init-plugin:
[INF] Invoking plugin (/Applications/XAMPP/xamppfiles/htdocs/masteringextjs/packages/masteringextjs-theme/.sencha/package/plugin.xml) - supported targets: -before-pkg-build
[INF]
[INF] -before-pkg-build:
[INF] Invoking plugin (/Applications/XAMPP/xamppfiles/htdocs/masteringextjs/packages/masteringextjs-theme/.sencha/package/plugin.xml) - supported targets: pkg-build
[INF]
[INF] cmd-root-plugin.init-properties:
[INF]
[INF] init-properties:
[INF]
[INF] init-sencha-command:
[INF]
[INF] init:
[INF]
```

该命令在packages/masteringextjs-theme文件夹里创建了build目录，如下图所示：



build文件夹里有一个resources目录，在resources目录里我们可以找到一个masteringextjs-theme-all.css文件，该文件包含了在我们的自定义主题（目前还没有，后续会实现）中进行样式设计的所有组件的所有样式。即使创建了一个完整的主题（对所有组件进行样式设计），也并非一定会在应用程序中用到所有的组件。Sencha Cmd能够帮我们过滤并创建一个CSS文件，其中只包含需要用到的组件。因此，我们不需要在应用程序中手动包含masteringextjs-theme-all.css文件。

现在，我们对项目进行相关设置，使其能够应用自定义主题。在项目文件夹中，有一个隐藏的文件夹叫.sencha，在这个文件夹里，有一个app文件夹及一个sencha.cfg文件，如下图所示：



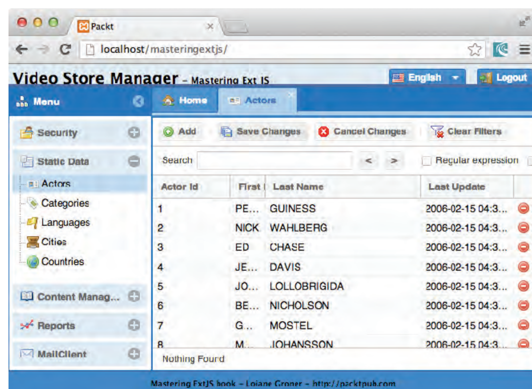
文件里包含了Sencha Cmd用到的项目属性配置，因此更改这些配置项时一定要小心。定位到`app.theme`属性，并更改其值为我们创建的主题名称（`masteringextjs-theme`），如下图所示：

```
32  # The name of the package containing the theme scss for the app
33  app.theme=masteringextjs-theme
```

接下来需要在项目中应用这些更改，打开终端应用程序，进入应用根目录并执行以下两条命令：

```
sencha ant clean
sencha app build
```

如果我们试着在浏览器里打开应用程序，将发现应用程序已经应用了`Neptune`主题：



不过，到目前为止仍没有什么改变。让我们马上开始自定义主题吧！先回到packages/masteringextjs-theme文件夹，在sass/var文件夹里创建一个新文件Component.scss，为其添加以下内容：

```
$base-color: #317040 !default;
```

我们通过这行代码声明了一个名为`$base-color`的Sass变量，其值为绿色十六进制颜色码（`#317040`）。该设置把主题的基本颜色由原先的蓝色更改为绿色。我们来应用一下这个更改看看效果如何。

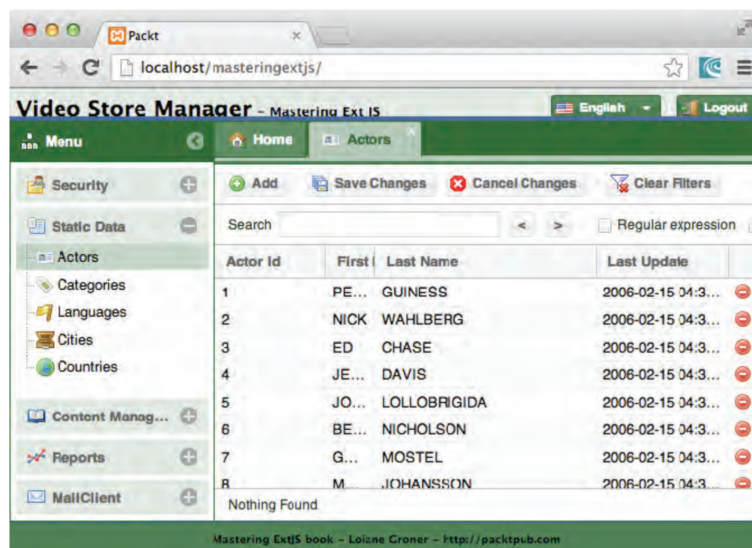
打开终端应用程序，进入 packages/masteringextjs-theme目录，执行以下命令：

```
sencha package build
```

之后，切换到项目根目录下，并执行以下命令：

```
sencha app build
```

打开浏览器，显示效果如下图所示：



我们可以继续为自定义主题添加更多的样式。不管我们做了什么改变并希望看到何种效果，都需要按照前面的做法，执行`sencha package build`和`sencha app build`这两条命令。

Ext JS 4.2与Ext JS 4早期版本在生成自定义主题方面有着很大的差异。之前的版本里，我们会使用Compass编译（`compass compile`），并且手工处理文件的所有创建及编译工作。现在，Sencha Cmd已能够帮我们做这些事情，我们只需聚焦在希望创建的Sass样式代码上。但是，新方式比老方式更耗时。

Ext JS文档非常完整。如果你想自定义一个特定的组件，你可以在文档中找到所有Sass的CSS变量以及组件使用的CSS混入类。





若希望了解更多关于Ext JS应用主题的信息，请访问：http://docs.sencha.com/extjs/4.2.0/#!/guide/theming。同时，强烈建议大家掌握Sass（http://sass-lang.com/）和Compass（http://compass-style.org/）。

10.3　为产品发布打包应用

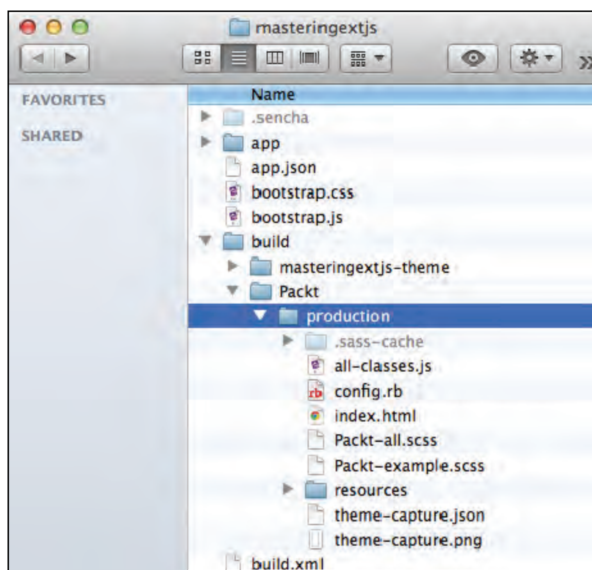
我们将主题创建好了。现在剩下的唯一工作就是构造产品并将其发布到生产环境的Web服务器上。我们依旧通过Sencha Cmd来完成这件工作。

打开终端应用程序，切换至应用系统根目录并执行以下命令：

```
sencha app build
```

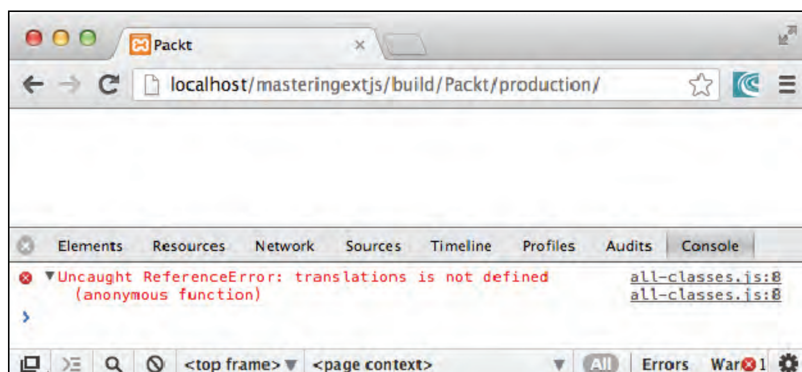
```
masteringextjs — bash — 80×24
loiane:~ loiane$ cd /Applications/XAMPP/xamppfiles/htdocs/masteringextjs
loiane:masteringextjs loiane$ sencha app build
Sencha Cmd v3.1.0.256
[INF] Including theme package masteringextjs-theme for app.theme=masteringextjs-
theme build
[INF]
[INF] init-plugin:
[INF]
[INF] init-plugin:
[INF] Invoking plugin (/Applications/XAMPP/xamppfiles/htdocs/masteringextjs/.sen
cha/app/plugin.xml) - supported targets: -before-app-build
[INF]
[INF] -before-app-build:
[INF] Invoking plugin (/Applications/XAMPP/xamppfiles/htdocs/masteringextjs/.sen
cha/app/plugin.xml) - supported targets: app-build
[INF]
[INF] cmd-root-plugin.init-properties:
[INF]
[INF] init-properties:
[INF]
[INF] init-sencha-command:
[INF]
[INF] init:
[INF]
```

一旦命令执行完毕，将创建一个新目录build/NameofTheApp/production。由于我们的应用程序命名空间为Packt，所以创建的目录就是build/Packt/production，如下图所示：



这条命令的作用是获取我们开发的所有代码（app目录中）以及应用程序所需的相关Ext JS代码，并将其放入all-classes.js文件中。之后，通过YUI Compressor工具（YUI压缩工具），Sencha Cmd压缩并混淆JavaScript代码；这样用户就只需加载非常小的JavaScript代码。另外，Sencha Cmd将鉴别应用程序使用到的组件，过滤掉不需要的CSS样式设置，放在resources/Packt-all.css文件里。我们所有的自定义图标也将从开发环境复制到生产文件夹中（同样放在resources文件夹中）。

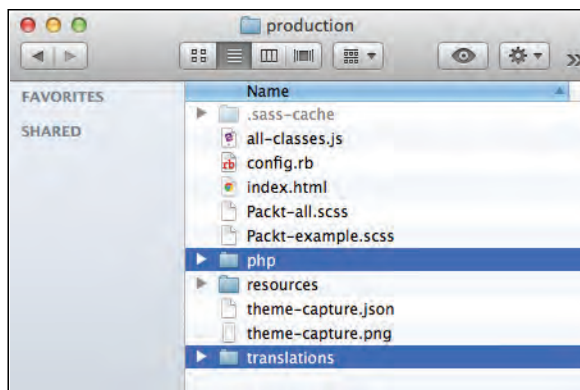
接下来要确保产品构造能如预期一样正常工作。我们通过http://localhost/masteringextjs访问开发环境，而要测试产品化的构造效果就得访问http://localhost/masteringextjs/build/Packt/production。在测试过程中会发现结果并不正常，错误如下图所示：



因此，首先打开build/Packt/production/index.html文件并修改它。我们需要添加语言转换文件以及app.css文件（该样式文件是在第1章创建图标时一同创建的），最终版本如下：

```
<!DOCTYPE HTML>
<html>
<head>
    <meta charset="UTF-8">
    <title>Packt</title>
<link rel="stylesheet" href="resources/Packt-all.css"/>
<link rel="stylesheet" href="resources/css/app.css">
<script src="translations/locale.js"></script>
<script type="text/javascript" src="all-classes.js"></script>
</head>
<body></body>
</html>
```

接下来复制translations文件夹和php文件夹到production文件夹下，如下图所示：

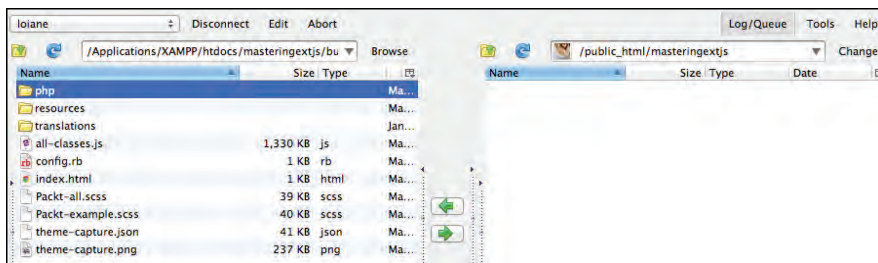


再次测试，结果就正常了。

10.3.1　发布成产品的内容

别忘了我们的app文件夹以及所有代码都是在开发环境中创建开发的。产品文件夹中所有的这些开发代码都要发布成产品代码。

那么现在就来发布应用程序吧。先把masteringextjs/build/Packt/production的内容移到Web服务器的文件夹下，如下图所示：



10.3.2　产品化的优点

产品化的好处是什么？我们能只发布开发代码吗？可以把开发代码当成产品代码来发布，但我们不推荐这种方式。通过产品化构造过程，文件被压缩了，加载文件时的性能就会提高。

比如，我们来做个测试：在浏览器里打开应用程序，登录，然后打开静态数据模块的Actors（演员）界面。

在Google Developer Tools（或Firebug）中可以看到开发代码的输出结果，如下图所示：

Elements　Resources　Network　Sources　Timeline　Profiles　Audits　Console

Name Path	Method	Status Text	Type	Initiator	Size Content	Time Latency
data:image/gif;base...	GET	Success	image/gif	Script	(from cache)	0 ms
data:image/gif;base...	GET	Success	image/gif	ext-dev.js:22614 Script	(from cache)	0 ms 0 ms

911 requests ❘ 4.8 MB transferred ❘ 13.92 s (onload: 3.29 s, DOMContentLoaded: 405 ms)

All　Documents　Stylesheets　Images　Scripts　XHR　Fonts　WebSockets　Other

应用程序产生了911个请求，花费13.92秒将4.8 M的结果数据传给用户。这数据太大了，要传4.8 M的数据给用户显然无法令人接受。

来看看产品构造后的结果：

Elements　Resources　Network　Sources　Timeline　Profiles　Audits　Console

Name Path	Method	Status Text	Type	Initiator	Size Content	Time Latency
data:image/gif;base...	GET	Success	image/gif	Script	(from cache)	0 m
data:image/gif;base...	GET	Success	image/gif	all-classes.js:1 Script	(from cache)	0 m 0 m

450 requests ❘ 87.9 KB transferred ❘ 32.09 s (onload: 566 ms, DOMContentLoaded: 566 ms)

All　Documents　Stylesheets　Images　Scripts　XHR　Fonts　WebSockets　Other

应用程序产生了450个请求（依然很大）以及87.9 KB的传输量。450个请求数仍比较大，这是因为我们要呈现大量图标。在第12章里，我们将学到怎样借助其他工具的帮助来降低这个数量。但现在最重要的改变是传输数据量的大小，从4.8 M变为了87.6 KB，这是多么大的一个提升啊！

另一个需要的关注的是已加载的文件。在开发环境中，可以看到被浏览器加载的每个Ext JS类：

Elements　Resources　Network　Sources　Timeline　Profiles　Audits　Console

Name Path	Method	Status Text	Type	Initiator	Size Content	Time Latency	Timeline
Layout.js?_dc=1367890291845 /masteringextjs/ext/src/layout	GET	200 OK	applicatio...	ext-dev.js:10412 Script	20.3 KB 19.9 KB	944 ms 797 ms	
ElementContainer.js?_dc=136789029 /masteringextjs/ext/src/util	GET	200 OK	applicatio...	ext-dev.js:10412 Script	11.1 KB 10.8 KB	943 ms 807 ms	
None.js?_dc=1367890292082 /masteringextjs/ext/src/layout/contair	GET	200 OK	applicatio...	ext-dev.js:10412 Script	4.0 KB 3.7 KB	641 ms 583 ms	
Menu.js?_dc=1367890292083 /masteringextjs/ext/src/layout/contair	GET	200 OK	applicatio...	ext-dev.js:10412 Script	15.1 KB 14.7 KB	715 ms 644 ms	

光是渲染登录界面就有400多个JavaScript文件被加载。

10

接下来观察一下产品构造后的情况：

Elements　Resources　Network　Sources　Timeline　Profiles　Audits　Console

Name Path	Method	Status Text	Type	Initiator	Size Content	Time Latency	Timeline
locale.js /masteringextjs/build/Packt/productio	GET	200 OK	applicatio...	localhost:8 Parser	712 B 335 B	9 ms 6 ms	
all-classes.js /masteringextjs/build/Packt/productio	GET	304 Not Modifiec	applicatio...	localhost:8 Parser	263 B 1.3 MB	8 ms 6 ms	
en.js /masteringextjs/build/Packt/productio	GET	304 Not Modifiec	applicatio...	locale.js:5 Script	285 B 793 B	6 ms 5 ms	
ext-lang-en.js /masteringextjs/build/Packt/productio	GET	200 OK	applicatio...	locale.js:6 Script	11.1 KB 10.7 KB	45 ms 3 ms	

只有4个JavaScript文件被加载，差别实在太大了！

因此，出于性能的考虑，应该始终构造并发布产品。开发代码只是用于开发目的。另外，还可能出于测试目的实现测试构造，我们将在第12章进一步讨论这个话题。

10.4　从 Web 到桌面：Sencha Desktop Packager

对Ext JS开发专家而言，产品部署就是意味着发布Ext JS代码和服务器端代码到Web服务器上；至于使用PHP、Java、.NET、Ruby或其他何种语言都没有问题。所有代码——前端（Ext JS）代码和后端（如PHP）代码，都需要发布到Web服务器上。

但还有一种发布Ext JS代码给用户的方式：发布成桌面应用程序。本书并不讨论使用Java桌面开发技术（比如Swing）或C/C++技术开发项目，我们只讨论用HTML、CSS和JavaScript开发原生应用（Mac OS、Linux和Windows上），当然，使用的是我们喜欢的框架——Ext JS。

市面上有一些工具能够帮助我们构建Ext JS桌面应用程序，但本书将介绍Sencha的另一个工具——Sencha Desktop Packager。Sencha Desktop Packager是Sencha的付费工具，但可以下载并体验一下其试用版本。下载地址为http://www.sencha.com/products/desktop-packager/。

接下来实现部署在Mac OS、Linux以及Windows平台上的应用程序原生桌面版本。另外，我们还需要知道Sencha Desktop Packager能否满足生成应用代码包的需要，以及是否还需做些额外的工作。下面，我们一步步来完成这些工作。

10.4.1　安装Sencha Desktop Packager

下载Sencha Desktop Packager并解压到我们选择的目录中。

下一步添加 Sencha Desktop Packager到操作系统的`PATH`（路径）环境变量中。

1. Mac OS和Linux操作系统

在Mac OS和Linux操作系统中，可以通过终端应用程序来完成此操作。假设我们使用

/Users/loiane/bin/Sencha/SenchaDesktopPackager作为Sencha Desktop Packager的目录。

要设置PATH，只需使用下面这条命令：

```
export PATH=$PATH:/Users/loiane/bin/Sencha/SenchaDesktopPackager
```

之后，我们可以打印出PATH变量（echo $PATH）以检查PATH是否设置成功。最后，通过ionpackage命令测试是否安装正常：

```
loiane — bash — 83x24
loiane:~ loiane$ export PATH=$PATH:/Users/loiane/bin/Sencha/SenchaDesktopPackager
loiane:~ loiane$ echo $PATH
/Users/loiane/bin/Sencha/Cmd/3.1.0.256:/Users/loiane/bin/Sencha/Cmd/3.1.0.130:/Users/loiane/bin/Sencha/Cmd/3.0.0.250:/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin:/usr/X11/bin:/usr/local/git/bin:/Users/loiane/bin/Sencha/SenchaDesktopPackager
loiane:~ loiane$ ionpackage
Sencha Desktop Packager 1.1.1.398

Usage: ionpackage [options] configFile.json

Options:
   --activate       Activate a full license.
   --overwrite      Force overwriting the target path (if it already exists).
   --version        Show the version.
   -h or --help     Display this usage.

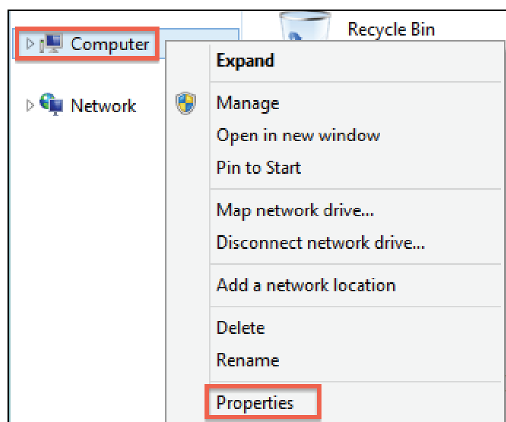
For more information, see the documentation.

loiane:~ loiane$
```

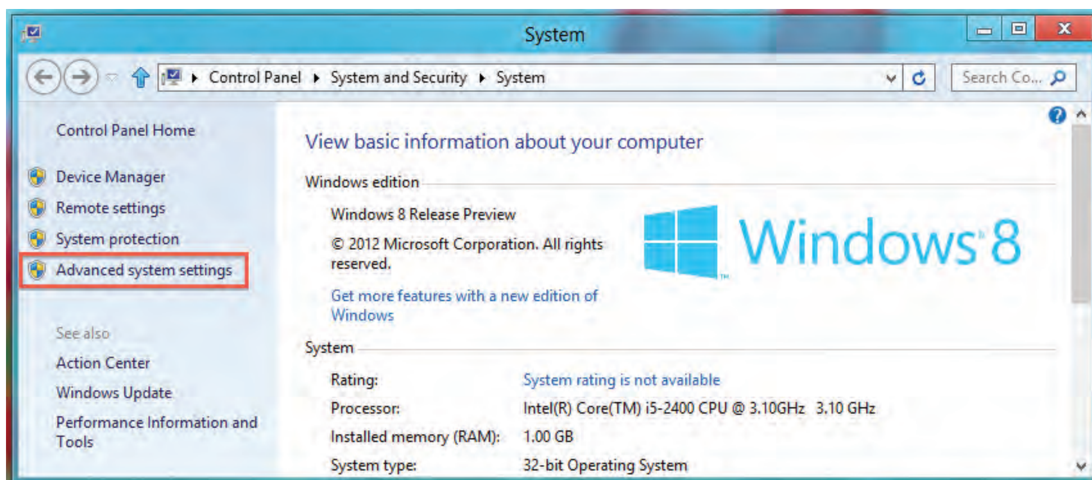
2. Windows操作系统

如果使用Windows操作系统，就需要通过另一种方式设置环境变量。我们用C:/SenchaDesktopPackager作为Sencha Desktop Packager的目录。下图使用的是Windows 8操作系统，但步骤跟在Windows 7中是一样的，Windows XP也类似。

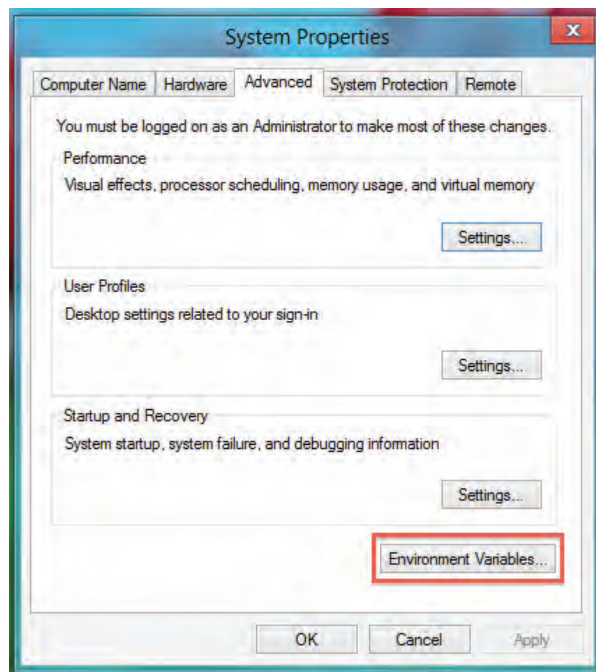
定位到计算机的图标Computer，右击选择Properties（属性）：



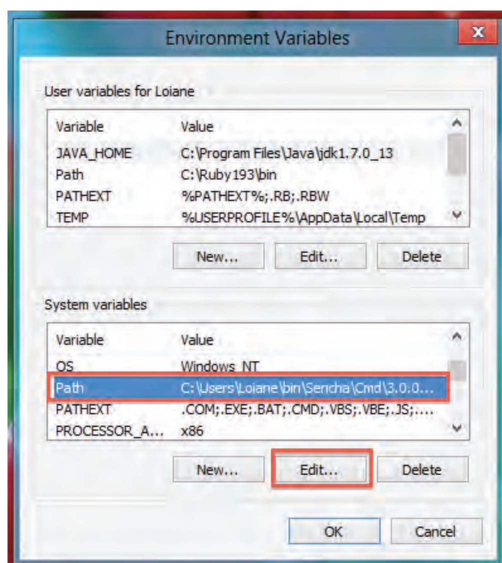
点击Advanced system settings（高级系统设置）：



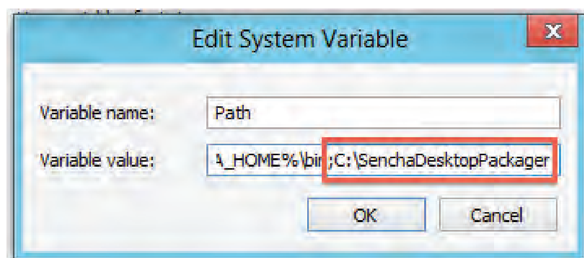
点击Environment Variables...（环境变量）：



在System variables（系统变量）框里，定位到Path并点击Edit...（编辑）：



在最后添加;C:\SenchaDesktopPackager并点击OK保存修改：



打开终端应用程序，敲入ionpackage命令测试安装是否成功，如下图所示：

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 6.2.8400]
(c) 2012 Microsoft Corporation. All rights reserved.

C:\Users\Loiane>ionpackage
Sencha Desktop Packager 1.1.1.398

Usage: ionpackage [options] configFile.json

Options:
    --activate        Activate a full license.
    --overwrite       Force overwriting the target path (if it already exists).
    --version         Show the version.
    -h or --help      Display this usage.

For more information, see the documentation.

C:\Users\Loiane>_
```

10

这样，Sencha Desktop Packager就安装完成了！

10.4.2　应用打包

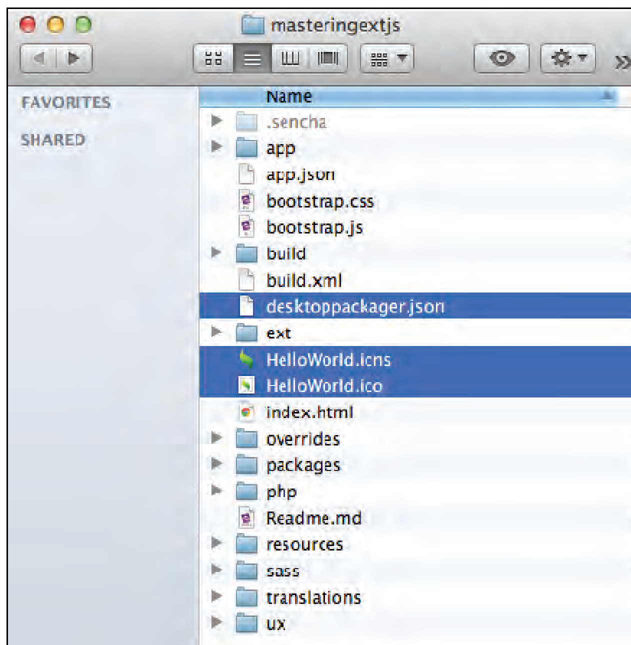
任务之一是产品部署准备，前面我们已经完成了。

在项目文件夹中，我们需要创建一个包含配置信息的JSON格式新文件desktoppackager.json：

```
{
    "applicationName"       : "Mastering Ext JS",
    "applicationIconPaths" : ["HelloWorld.ico", "HelloWorld.icns"],
    "versionString"        : "1.0",
    "outputPath"           : "build/Packt/package/",
    "webAppPath"           : "build/Packt/production/",
    "settings"             : {
        "mainWindow" : {
            "autoShow" : true
        }
    }
}
```

还需要一个图标用以表示应用程序。我们使用HelloWorld示例的HelloWorld.icns和HelloWorld.ico图标，可以在Sencha Desktop Packager的示例文件夹中找到它们。

添加三个新文件之后，我们的应用程序如下图所示：



之后，打开终端应用程序，切换到应用程序目录，并执行以下命令：

```
ionpackage desktoppackager.json
```

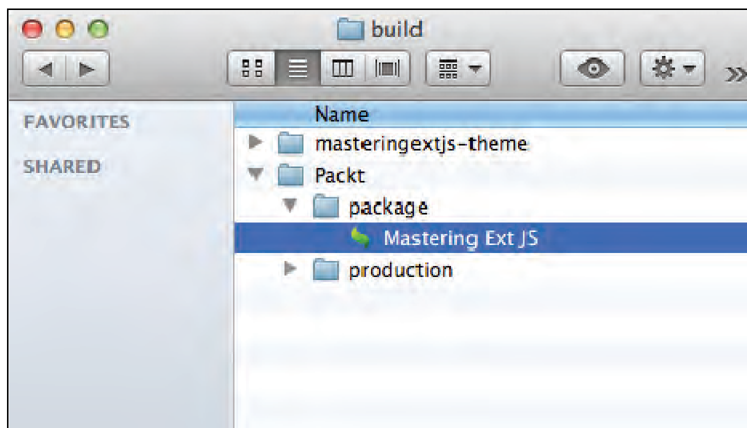
输出效果如下图所示：

```
masteringextjs — bash — 80×19
loiane:~ loiane$ cd /Applications/XAMPP/xamppfiles/htdocs/masteringextjs
loiane:masteringextjs loiane$ ionpackage desktoppackager.json
 -- License is valid
Packing from configuration file desktoppackager.json...
Searching for files. Please wait...
Total of 1581 files to be packaged.
Target package file /Applications/XAMPP/xamppfiles/htdocs/masteringextjs/build/P
ackt/package/app.ion
Package created at /Applications/XAMPP/xamppfiles/htdocs/masteringextjs/build/Pa
ckt/package/
Package /Applications/XAMPP/xamppfiles/htdocs/masteringextjs/build/Packt/package
/app.ion (14767937 bytes) is verified. Everything looks good.
Encrypting package with basic encryption...
Package encrypted successfully.
Extracting run-time to /Applications/XAMPP/xamppfiles/htdocs/masteringextjs/buil
d/Packt/package/
Run-time is extracted.

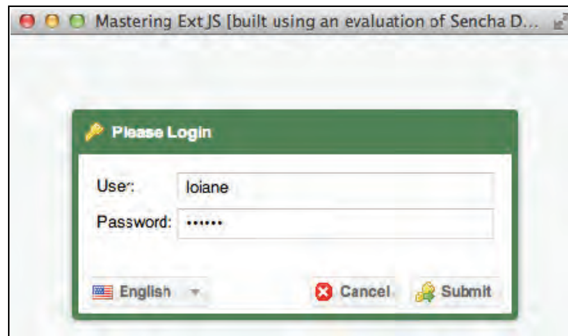
---------------------------------------------------
```

第一次使用Sencha Desktop Packager时，将提示你输入Sencha用户ID和密码。用户ID和密码跟你在Sencha论坛中注册的一致。

命令执行完毕后，打开build目录，可以看到刚才创建的新目录（Packt/package），其中，有一个名为`Mastering Ext JS`的原生应用程序（`.app`为Mac OS的，`.exe`是Windows的）：



如果现在运行程序，仍会看到一些错误信息。但我们已可以看到如下所示的登录界面（注意此处使用的是我们的自定义主题）：



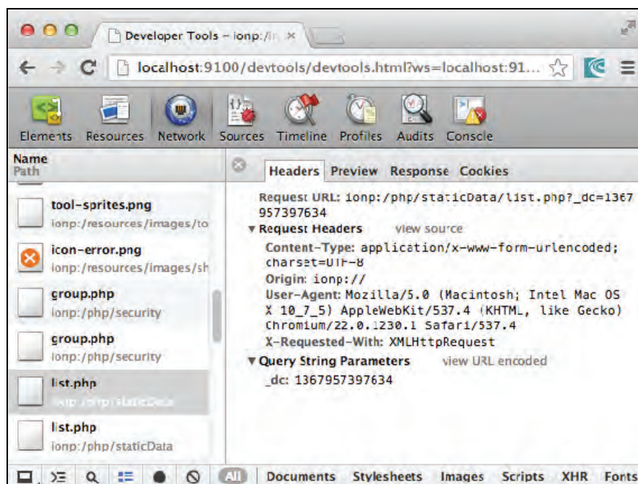
Sencha Desktop Packager会为我们使用的操作系统生成对应的原生应用程序。比如：如果在Mac OS上执行`ionpackage`，生成的是Mac OS上的原生应用程序。如果是在Linux上执行，则生成Linux的原生应用程序；在Windows上执行，则生成的就是Windows的原生应用程序。

10.4.3　服务器端代码调整

为什么会有错误信息出现呢？由于是原生应用，我们怎样才能知道发生了什么错误呢？能够像在浏览器里运行那样进行调试吗？要达成此目标，就需要在`desktoppackager.json`的`settings`属性配置项里添加以下配置信息：

```
"remoteDebuggingPort": 9100
```

之后，再次执行`ionpackage desktoppackager.json`命令，等待新的可执行文件生成。打开该文件，同时也打开一个`Webkit`内核的浏览器（如Chrome），打开http://localhost:9100链接。浏览器将列出应用程序中各个文件的名称，当我们在上面点击时，Google Developer Tools的操作选项就会变得可用：



观察一下就会注意到Sencha Desktop Packager同时还打包了PHP代码，应用程序试图把PHP文件当成JavaScript文件来访问。由于Sencha Desktop Packager不支持PHP代码，只能够处理HTML、CSS和JavaScript代码，因此这时候就会无法正常工作。

我们的服务器端代码，不管是用何种语言（PHP、Java或.NET）开发的，都必须发布到Web服务器上供桌面应用访问。

因此进行第一步处理时，我们需要在所有的存储器、代理以及Ajax请求中添加完整的URL。

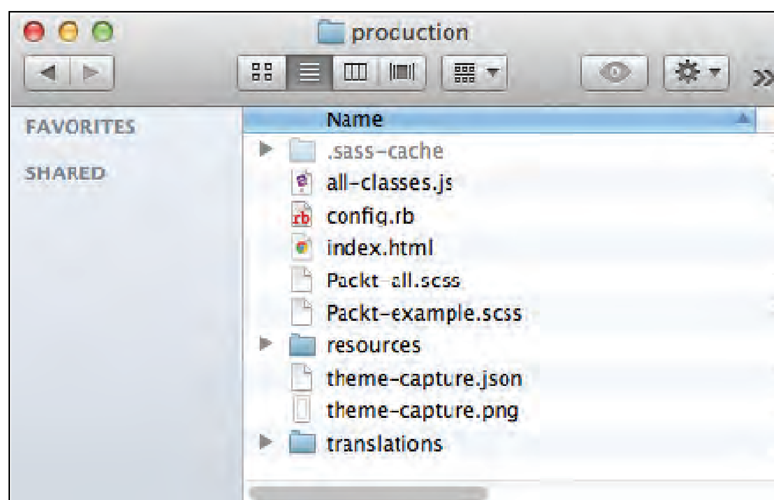
以菜单存储器为例。在原来的URL之前添加`http://localhost/masteringextjs`，结果如下：

```
url: 'http://localhost/masteringextjs/php/menu.php'
```

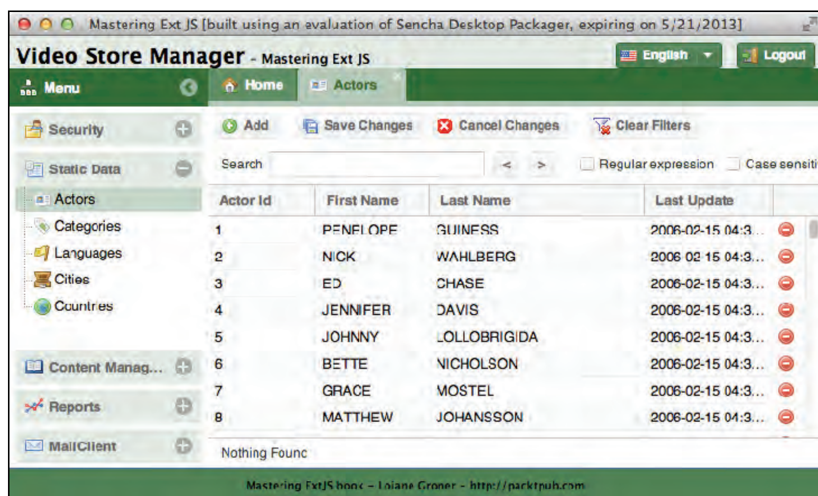
现在，搜索所有的存储器、代理以及Ajax请求（在控制器内部）并执行相同操作。所有URL的处理都必须采用上述菜单存储器的操作。

完成所有的改动后，重新做一次产品构造（别忘了再次修改build/Packt/production目录下的index.html文件；或者先把index.html复制到某处，待产品构造过程结束后再替换回来。每次重新进行产品构造，index.html都会被覆盖）。现在来验证一下能否运行：从产品构造中移除php文件夹并测试（http://localhost/masteringextjs/build/Packt/production/）。运行正常了，没有任何错误。

Sencha Desktop Packager产品构造的目录如下图所示（除了php文件夹被移除，其他的文件夹和文件都跟前一次构造时一样）：



现在我们需要再次运行`ionpackage desktoppackager.json`命令。接下来再次测试我们的原生应用程序。结果将与在Web版本上运行一样，不同之处只在于它是原生应用程序：



如果我们打包Windows版本的应用程序，结果也一致，但运行的是.exe文件，应用程序看起来就是Windows程序风格。在Linux上也一样。

Ajax、JSONP与CORS的比较

必须明白一件很重要的事情：当前，我们的服务器端代码发布在本地，应用程序也运行在本地。

记住以下这一点是很重要的：我们在应用程序中一直使用Ajax请求，而Ajax只允许域内请求，不允许跨域请求。

假设我们要在几个不同的域（domainA、domainB、domainC，等等）为用户发布应用程序，而服务器端代码又发布在packt.com，那结果会怎样呢？肯定会出错，因为Ajax无法实现跨域请求。

这时候我们可以使用JSONP，但得修改服务器端代码以返回JSONP回调参数，并且JSONP只适用于GET请求，这就意味着我们只能取回信息，无法使用POST、PUT以及DELETE请求。

还好有第三个选择方案，就是跨域资源共享（Cross-Origin Resource Sharing，CORS）。举个例子，如果在服务器端代码中启用CORS，就可以实现从domainA到packt.com的Ajax请求。这时候，服务器端代码的改动是最小的。

如在PHP中，我们只需在所有PHP文件（每个文件开头处）中添加以下代码：

```
<?php header("Access-Control-Allow-Origin: *");
```

问题就解决了。

我们同时还需要调整一下desktoppackager.json文件的内容，比如添加一个允许跨站点访问的特定配置项。

```
{
    "applicationName"      : "Mastering Ext JS",
    "applicationIconPaths" : ["HelloWorld.ico", "HelloWorld.icns"],
    "versionString"        : "1.0",
    "outputPath"           : "build/Packt/package/",
    "webAppPath"           : "build/Packt/production/",
    "settings"             : {
        "remoteDebuggingPort": 9100,
            "security": {
                "allowCrossSite": true
            },
        "mainWindow" : {
            "autoShow" : true
        }
    }
}
```



若希望了解更多关于Sencha Desktop Packager配置及其作用的信息，可访问http://docs.sencha.com/desktop-packager/1.1。

因此，如果我们要发布原生应用程序给不同的用户，就有可能需要在代码中启用CORS。如果在一个服务器发布Ext JS代码，而在另一个不同域的服务器上发布服务器端代码，也需要做同样处理。



若希望了解更多关于CORS的信息，包括怎样在不同服务器端语言中启用它、浏览器支持以及其他信息等，可访问http://enable-cors.org/。

10.5　小结

本章学习了如何使用新的Ext JS 4.2主题引擎创建一个新主题，了解了产品构造的重要意义，掌握了产品构造的方法，理清了开发环境文件与生产环境文件之间的差异。

我们还了解了Sencha Desktop Packager，以及怎样把我们的Ext JS程序打包成可以在Mac OS、Linux以及Windows上执行的原生应用程序。在这个过程中，我们了解到打包应用程序并不是件简单的事情，还需要做一些改动。本章还讲解了在满足跨域访问的情况下，发布应用程序的可选解决方案。

下一章我们将学习如何使用Ext JS创建一个完整的WordPress主题。

第 11 章

创建WordPress主题

毫无疑问，Ext JS是个优秀的JavaScript框架，不仅能够用来开发CRUD风格的应用程序，同时还能用来开发如跟踪展示股票价格这样的实时应用程序。当然，Ext JS还可以实现你能想到的其他类型的应用程序。本章将用Ext JS创建一个WordPress主题。如果你还不熟悉WordPress，你就记住它是个可以用来创建漂亮网站或博客的Web应用发布平台。换言之，WordPress是一个博客内容管理软件。你将注意到，本章使用的开发方法完全不同于前几章中用到的开发方法。

本章主要包括以下内容：

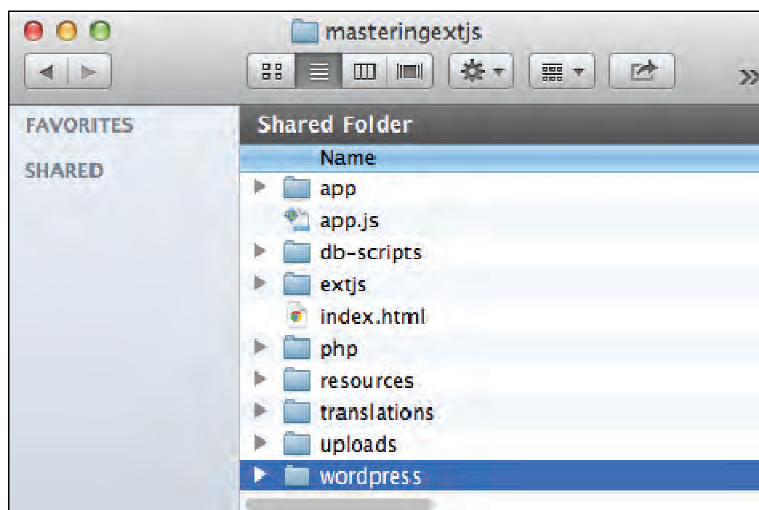
- 组织主题结构；
- 构建头部和页脚；
- 构建主页面；
- 构建侧边栏；
- 构建单一文章页面；
- 构建单一页面。

11.1 安装 WordPress

在开始逐步讲解之前，需要安装WordPress。你可以使用已有博客的WordPress（比如http://loiane.com和http://loianegroner.com上用的WordPress），或者进行本地安装。

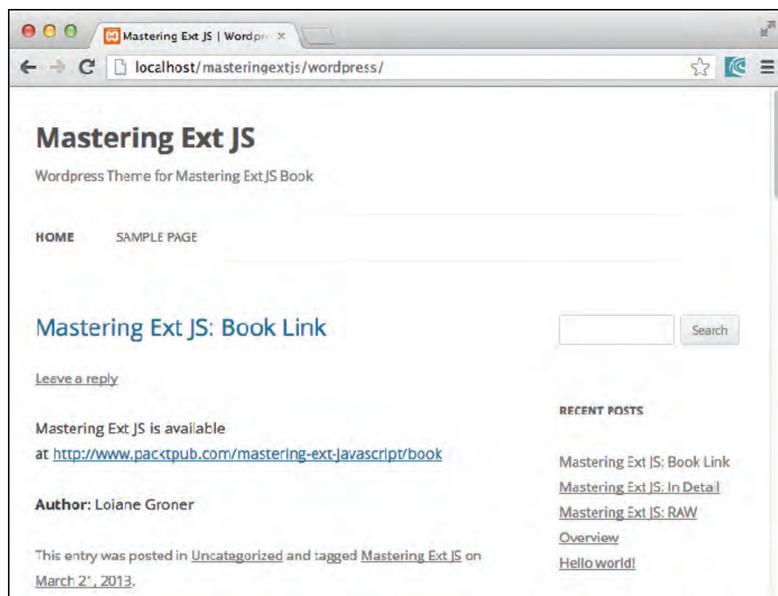
如果你不会安装WordPress，可以参考以下教程：http://codex.wordpress.org/Installing_WordPress。

出于测试目的，我们使用本地安装的方式。将WordPress安装在XAMPP htdocs文件夹中的masteringextjs文件夹中，如下图所示：

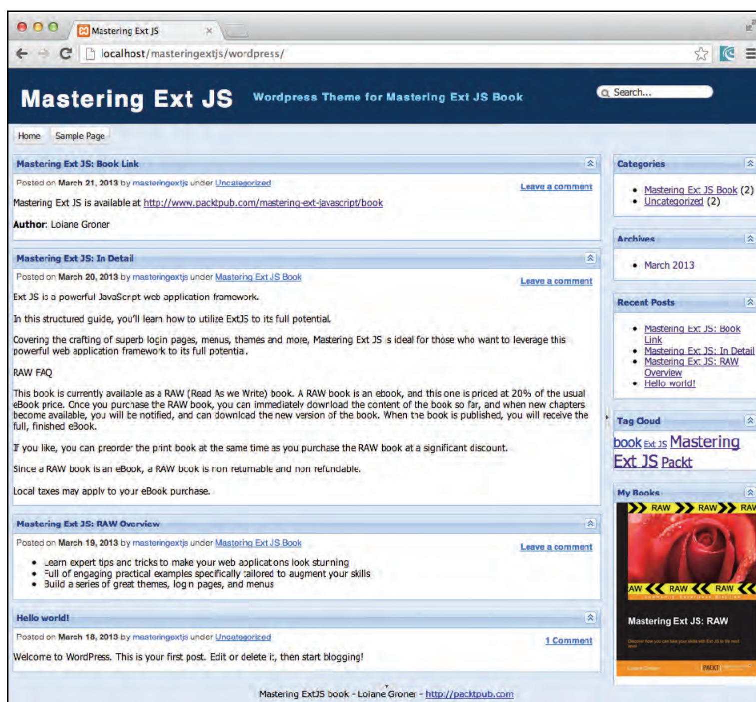


上图显示了出于测试目的，安装WordPress的一些信息。我们将基于此安装版本创建主题。

WordPress安装完毕后，通过浏览器可以看到以下效果（WordPress安装后使用的是默认主题）：



当用Ext JS为WordPress实现了主题后，最终效果图如下所示：



11.2　WordPress 主题简介

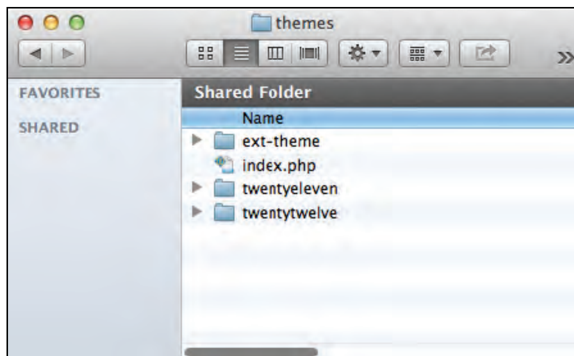
WordPress主题是WordPress如此流行的一个重要原因。在开发之前，我们需要了解一下WordPress主题的工作原理。当创建一个新的WordPress主题时，需要创建一些被WordPress自动组织和管理的文件。这些文件大多数都一目了然，但我们还是来浏览一下。

- header.php　加载博客时，WordPress将该页面转换为一个HTML页面，以便浏览器可以对其渲染呈现。header.php文件里包含了到`</head>`标签之前的主题设置代码。
- sidebar.php　这是个可以用来被WordPress的`get_sidebar()`函数调用的可选文件。我们可以在其中添加代码，用于渲染小部件以及主题的侧边栏（如果该部分存在的话）。
- footer.php　在此结束HTML代码以构建完整的主题。你也可以根据需要在这里呈现小部件。
- page.php　用于呈现单一页面。如关于页面和新闻页面都是单一页面。
- single.php　用于呈现单一博客文章，与page.php的代码很相似。
- index.php　当博客被渲染呈现时调用index.php。其包含并呈现文章、搜索结果、头部、页脚、侧边栏、出错消息，等等。
- functions.php　我们可根据实际需要，在此添加额外的主题函数。
- comments.php　在此呈现评论、引用以及评论提交表单。

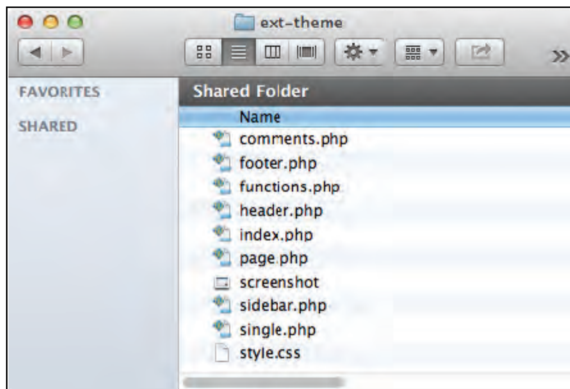
WordPress主题可以根据需要使用很多文件，但以上文件是创建主题时最常使用的。

11.3 组织主题结构

下面我们开始吧，先切换到主题目录来创建主题。主题目录为masteringextjs/wordpress/wp-content/themes（masteringextjs文件夹在XAMPP的htdocs文件夹内）。创建一个新文件夹ext-theme，如下图所示：



创建以下文件：comments.php、footer.php、functions.php、header.php、index.php、page.php、sidebar.php、single.php以及style.css等（这些文件可以是空的，现在只需创建它们即可），如下图所示：



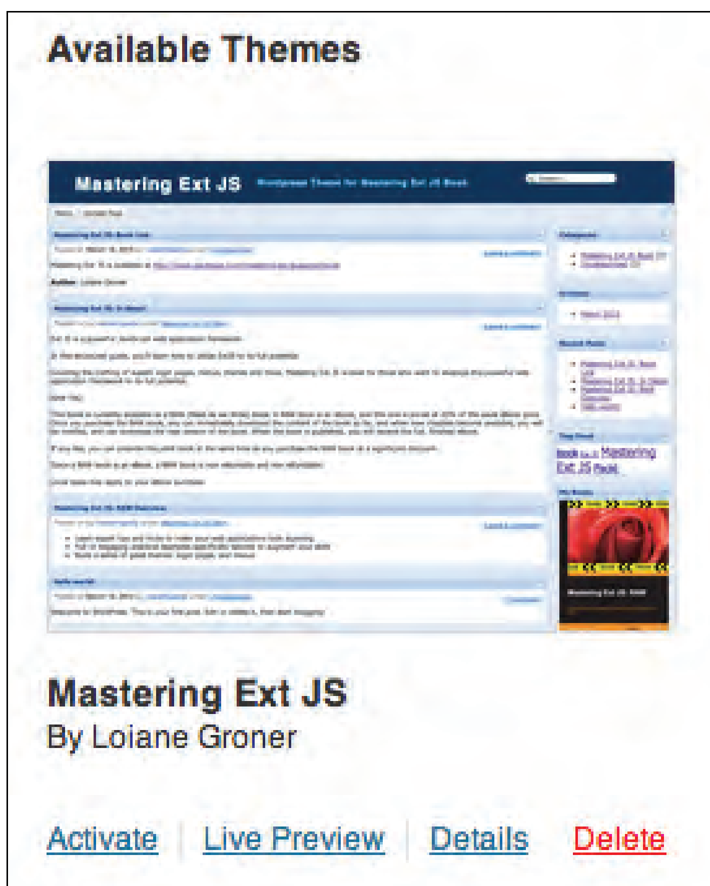
同时，在ext-theme文件夹中创建（或粘贴）一个名为screenshot.png的文件，我们将用它来表示主题。

现在我们搭建好了主题的结构，还需要让主题出现在WordPress站点控制板上。因此，要修改style.css文件，在其中添加以下内容：

```
/*
Theme Name: Mastering Ext JS
Theme URI: http://packtpub.com
Description: Wordpress theme example for Mastering Ext JS Book - http://packtpub.com

Author: Loiane Groner
Version: 1.0
Tags: minimalistic, simple, extjs, sidebar, elegant, masteringextjs
*/
```

这里，WordPress会尝试在主题目录中查找style.css文件，以提取有关主题的信息。如果在WordPress特性菜单中打开站点控制板，就可以看到我们的主题已出现在其中并可供激活（使用）了：



激活该主题并浏览博客，却发现博客是黑的。这是好事不用担心，因为这就意味着现在可以开始构建我们的主题了。

再来看看我们的主题截图，图中突出显示了主题最重要的部分：

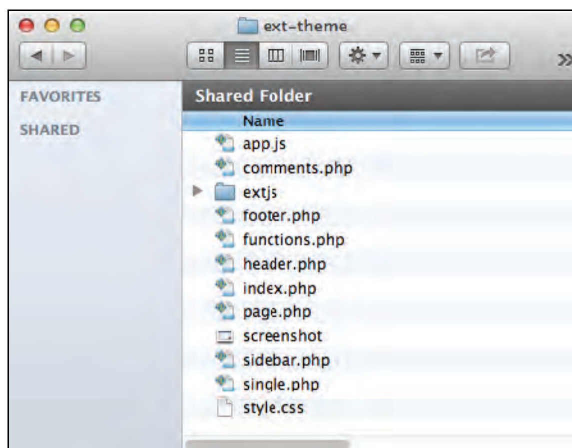


头部包含了包括标题、描述以及搜索区域。下一行是导航链接，以便读者可以在博客页面间导航。侧边栏包含了各种小部件，用以显示Categories（类别）、Archives（归档）、Recent Posts（最近文章）、Tag Cloud（标签云）以及自定义小部件（My Books，我的书籍）。页脚部分包含了版权信息。内容区域包含最近文章的列表。

接下来做点有趣的事情，开始编码实现我们的主题。

11.4　构建头部

使用Ext JS创建WordPress主题时，需要把Ext JS发行库安装到我们的主题目录中，并创建一个JavaScript文件，在其中将添加我们需要的所有Ext JS代码。接下来完成此项工作，复制Ext JS SDK到ext-theme文件夹中，同时创建一个新的JavaScript文件app.js：



现在，我们在header.php文件里添加以下代码：

```
<html>
<head>
<title><?php bloginfo('name'); ?> <?php wp_title(); ?></title>   // #1
<link rel="stylesheet" href="<?php bloginfo('stylesheet_directory'); ?>
/extjs/resources/css/ext-all.css" type="text/css"/> // #2
<link rel="stylesheet" href="<?php bloginfo('stylesheet_url'); ?>"/> // #3
<script src="<?php bloginfo('stylesheet_directory'); ?>/extjs/ext-all.js"></script>
// #4
<script src="<?php bloginfo('stylesheet_directory'); ?>/app.js"></script> // #5
</head>
```

WordPress有一个叫bloginfo的函数，允许我们获取博客的所有信息，如博客名、描述、主题目录、样式表URL、站点URL等。如你所见，我们在上述代码的5个不同地方使用了该函数。若希望了解关于该函数的更多信息，请访问：http://codex.wordpress.org/Function_Reference/bloginfo。

title标签显示博客名bloginfo('name')，同时还通过wp_title函数显示当前页名称（#1）。例如：如果我们打开Hello world!文章，标题将显示：Mastering Ext JS >> Hello world!。

接下来导入Ext JS文件：ext-all.css（#2）、ext-all.js（#4）以及app.js（#5）。要达成此目的，可以使用bloginfo('stylesheet_directory')函数获取当前主题目录，并结合每个文件的路径形成一个完整路径。



ext-all.js文件里包含了Ext JS框架的完整源代码。因此，这同前面章节里的处理方式不同。

最后，需要导入style.css文件。通过bloginfo('stylesheet_url')函数实现该操作（#3）。

接下来，为header.php文件添加更多的代码：

```
<body>
<div id="headerCont" style="display:none;">
  <!-- content here #6 -->
</div><!-- headerCont -->
```

我们将在headerCont这个div标签里添加用来显示博客标题和描述的代码、搜索区域以及博客导航链接。由于我们不希望内容仅显示为简单的HTML形式，因此将样式设为不显示该div标签，并打算用一个Ext JS组件来显示该内容。

用以下代码替换注释#6：

```
<div id="top-bar-tile">
<div id="top-bar-content">
  <h1><a href="<?php bloginfo('url'); ?>">
```

```
    <?php bloginfo('name'); ?> <!-- #7 -->
  </a></h1>
  <span class="slogan">
    <?php bloginfo('description'); ?> <!-- #8 -->
  </span>
  <div id="search-box"> <!-- #9 -->
    <form method="get" id="searchform" action="" >
      <input type="text" value="Search..."
      onfocus="if(this.value == this.defaultValue) this.value = ''"
      name="s" id="s" />
    </form>
  </div><!-- search-box -->
</div><!-- top-bar-content -->
</div><!-- top-bar-tile -->
```

显示博客名称（#7）及其描述（#8），还有search-box（#9）。

接下来，显示导航条，使用wp_nav_menu函数（http://codex.wordpress.org/Function_Reference/wp_nav_menu）在导航条上动态显示所有的博客页面（名称）：

```
<div id="links" style="display:none;">
   <?php wp_nav_menu(array(
   'menu' => 'mainnav',
   'menu_class' => 'nav-bar-content',
   'menu_id' => 'navigation',
   'container' => false,
   'theme_location' => 'primary-menu',
   'show_home' => '1')); ?>
</div><!-- links -->
```

最后，打开index.php文件并添加以下代码：

```
<?php get_header(); ?>
```

从PHP角度而言，这足以显示头部内容了。

创建Ext JS代码

现在我们要开始创建Ext JS代码了。创建一个如本章前面所描述的简单主题，并不需要太多的Ext JS代码。Ext JS端的实现很简单，主要工作都在于PHP及WordPress函数实现上。

因此，基本的实现思路是创建一个使用边界布局的视见区。在顶部区域放置头部，底部是页脚，右侧是侧边栏，中央区域是内容。不需要为这次的任务创建MVC应用程序；我们可以沿用Ext JS原有的实现方式，在Ext.onReady函数中编写代码。文章、小部件以及导航按钮等的所有实现都将通过原有且良好的DOM处理方式来动态完成。当然，你可以采取你希望的其他处理方式，但这次我们就采用这种方式。

在app.js文件中添加以下代码：

```
Ext.onReady(function() {
    Ext.create('Ext.container.Viewport', {
        layout: 'border',
        items: [{
            region: 'north',
            html: Ext.getDom('headerCont').innerHTML, // #1
            border: false,
            margins: '0 0 5 0',
            height: 100,
            dockedItems: [{
                xtype: 'toolbar',
                itemId: 'navToolbar', // #2
                dock: 'bottom',
                ui: 'footer'
            }]
        }, {
            region: 'center',
            xtype: 'container',  // #3
            autoScroll: true,
            styleHtmlContent: true,
            defaults: {
                xtype: 'panel', // #4
                padding: '5px',
                margins: '0 0 5 0',
                collapsible: true,
                styleHtmlContent: true,
                autoScroll: true
            }
        }]
    });
  // #5
});
```

顶部区域放置了一个面板，添加了headerContdiv标签的HTML内容，这也是设置该标签样式为style="display:none;"的原因（#1）。要显示导航链接，则需要添加一个navToolbar（#2）。过一会我们会实现相应的逻辑代码。

当使用边界布局创建一个容器时，中央区域是强制要求的，我们现在就来创建该区域。首先创建一个容器（#3），其中声明了子面板（#4）用来显示博客文章（一个子面板放置一篇文章）。

定义好视见区后，我们编写以下代码（#5）：

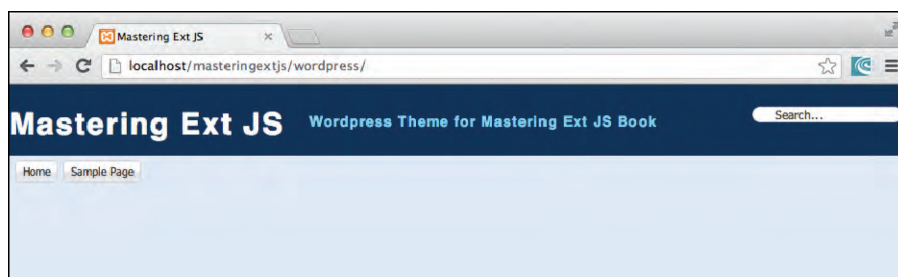
```
var buttons = Ext.get(Ext.getDom('links')).dom.children[0]; // #1
var list = Ext.get(buttons).child('ul').dom.children; // #2
var toolbar = Ext.ComponentQuery.query('toolbar#navToolbar')[0];
Ext.Array.each(list, function(li) { // #3
    toolbar.add({// #4
        text: Ext.get(li).dom.children[0].firstChild.data, // #5
        href: Ext.get(li).dom.children[0].href, // #6
        hrefTarget: '_self' // #7
    });
});
```

上述代码获取links div标签的第一个子元素（nav-bar-content div标签，#1）。之后，获取第一个ul标签子元素的子元素列表（#2）。接下来声明头部工具栏的引用。我们可以迭代（#3）li子元素列表并利用其内容创建工具栏的按钮（#4）。可以通过data属性获取按钮文本（#5）、通过href属性获取按钮链接（#6）。默认情况下，Link（链接）按钮将在新的浏览窗体中打开链接。由于我们希望在同一页面中打开链接，因此只用把hrefTarget设置为_self即可（#7）。

如何获取Ext JS对应的HTML代码呢？打开Firebug或Google Chrome developer tools，分析linksdiv标签创建的的HTML内容，将得到以下代码：

```
▼<div id="links" style="display:none;">
  ▼<div class="nav-bar-content">
    ▼<ul>
      ▶<li class="current_page_item">…</li>
      ▶<li class="page_item page-item-2">…</li>
     </ul>
   </div>
 </div>
 <!-- links -->
```

现在我们顺利完成了第一个实现（头部）。刷新博客页面，结果如下图所示：



11.5　构建页脚

下一步来构建页脚部分。在footer.php文件里添加以下代码：

```
<div id="footer" style="display:none;">
  <center>Mastering ExtJS book - Loiane Groner -
    <a href="http://packtpub.com">http://packtpub.com
  </a></center>
</div>
<?php wp_footer(); ?>
<?php get_sidebar(); ?>
</body>
</html>
```

和头部的实现一样，其中有一个div标签footer，同样不显示出来，其内容将在稍后实现

的容器中显示。div标签包含了一个简单的版权信息，你也可以添加一些你想要的内容，甚至是小部件。

下一步调用wp_footer函数。在WordPress模板里面，总是要在</body>关闭标签之前调用wp_footer函数。如果不调用该函数，就会中断一些插件的执行。该函数触发wp_footer动作，但并不会对我们主题的实现产生任何影响（我们仅需要按WordPress规定调用它即可）。

之后，我们调用get_sidebar函数。该函数将调用sidebar.php文件，相当于require("sidebar.php")。由于我们总是需要在页脚之后（wp_footer函数之后）加载sidebar.php文件，所以，就应该在footer.php文件的最后、</body>和</html>关闭标签之前调用get_sidebar函数。

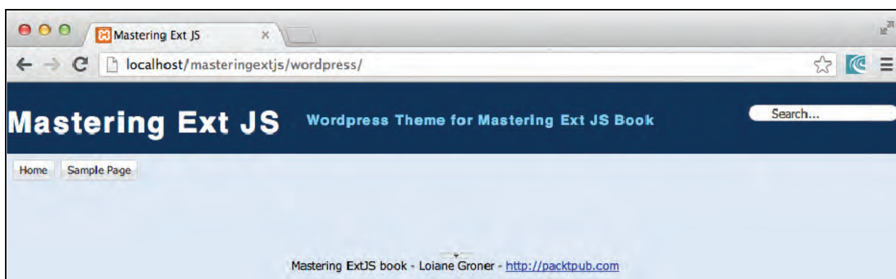
以上就是footer.php文件的内容。在index.php文件里也需要调用footer.php，因此，在头部调用（get_header()）之后添加以下内容：

```
<?php get_footer(); ?>
```

最后来实现app.js文件中的代码。在视见区里，添加以下子项内容：

```
{
    region: 'south',
    xtype: 'container',
    collapsible: true,
    html: Ext.getDom('footer').innerHTML,
    split: true,
    height: 25
}
```

页脚固定在页面底部，因此需要在视见区底部区域添加一个新容器，并使用footer标签的内容作为HTML内容。完成这些工作之后，刷新博客，结果如下图所示：



11.6　构建主页面

接下来创建主页面，即index.php文件。我们在其中已经添加了头部和页脚的调用代码，接下

来还需要添加显示博客文章列表的代码。因此，index.php文件实现如下：

```
<?php get_header(); ?>

<div id="main">
<div id="content">
  <div id="contentCont" style="display:none;">  <!-- #1 -->
  <?php while ( have_posts() ) : the_post(); ?>

    <div id='post' class="post"> <!-- #2 -->
      <div id="title" style="display:none;"><?php the_title(); ?></div> <!-- #3 -->
      <div class="post-details"> <!-- #4 -->
        <div class="post-details-left">
        Posted on <strong><?php the_date(); ?></strong> by <span class="author"><?php
the_author(); ?></span> under <span class="author"><?php the_category(',
'); ?></span>
        </div>
        <div class="post-details-right">
        <?php edit_post_link('Edit', '<span class="comment-count">  ' ,
'</span>'); ?><span class="comment-count"><?php comments_popup_link('Leave a
comment', '1 Comment', '% Comments'); ?></span>
        </div>
      </div>

      <?php if ( is_archive() || is_search() ) : // Only display excerpts for archives
and search. ?>
          <?php the_excerpt(); ?>
      <?php else : ?>
          <?php the_content('Read More'); ?>
      <?php endif; ?>

    </div><!-- post -->
  <?php endwhile; ?>
  </div><!-- contentCont -->
</div><!-- content -->
</div><!-- main -->

<?php get_footer(); ?>
```

为了更好地组织代码，内容区域用main div标签表示。其中有个id为contentCont的子div标签，同样有style="display:none;"（#1）这样的设置，其内容也不会显示出来，我们会再次获取该内容作为Ext JS容器的HTML内容。观察代码可知其中有个循环语句while (have_posts())。在WordPress主题中，我们称之为“The Loop”，其作用是获取所有WordPress数据库中的文章内容并将其显示在浏览器中，供用户浏览。

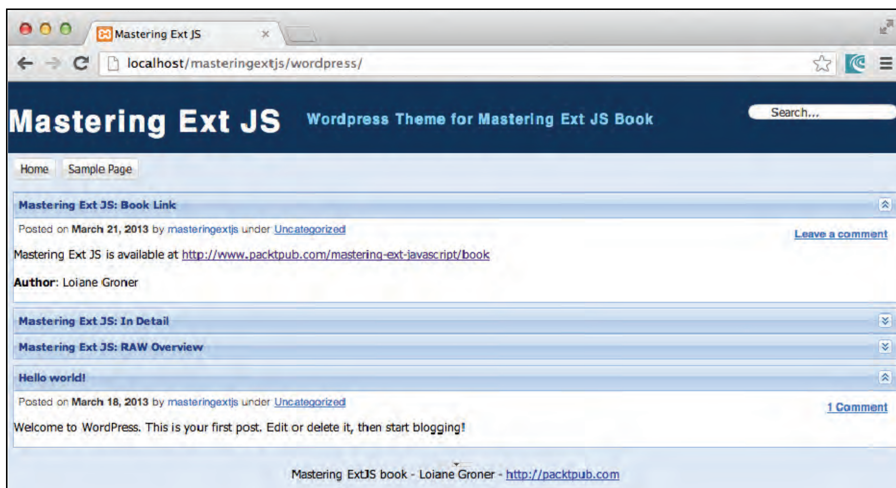
对于每篇文章，我们都将在id为post的div标签（#2）里显示其内容。这样的话，后续我们就可以操作DOM了。同时，我们也能够知道文章数量，并为每篇文章创建一个面板。之后，我们有一个div标签，其内容为文章标题（#3），使用它来设置面板标题。接下来，post-details div标签包含文章内容（#4），每个文章面板的HTML内容将用它来加载并显示。

接下来实现Ext JS代码，只需在视见区实现代码后添加以下内容即可：

```
var posts = Ext.get(Ext.getDom('contentCont')).dom.children;
var panel = Ext.ComponentQuery.query('container[region=center]')[0];
Ext.Array.each(posts, function(post) {
    panel.add({
        title: Ext.get(post).child('div#title').getHTML(),
        html: post.innerHTML
    });
});
```

变量posts是个包含了contentCont标签所有子元素的数组，是个post div标签的集合（文章集合）。接下来，我们获取在中央区域创建的容器的引用，以便为其添加面板子项。之后，通过循环把每篇文章（以面板形式）添加到容器中，获取title div标签以设置面板标题，并通过post div标签设置面板内容。

刷新博客，结果如下图所示：



中央区域的容器将包含一篇篇的文章，每篇文章都用一个单独的面板表示。

11.7　构建侧边栏

现在，我们来构建侧边栏。前面已经添加了调用代码（在footer.php文件中），接下来需要在sidebar.php文件里一步步添加实现代码。

首先，添加id为sidebar的div标签：

```
<div id="sidebar" style="display:none;">
<!-- #1 -->
```

```
</div>
```

注意上述代码中div标签的`style="display:none;"`设置。后面我们将再次对DOM进行处理，以便在Ext JS组件里显示该标签子元素。在注释位置#1中，将添加其他div标签，每个div标签代表侧边栏的一个小部件。

第一个要添加到侧边栏的小部件是带有各自文章数目的分类列表：

```
<div id="categoriesCont">
  <ul>
    <?php wp_list_cats('sort_column=name&optioncount=1&hierarchical=0'); ?>
  </ul>
</div>
```



wp_list_cats是一个WordPress函数，可用以获取分类列表。若希望了解更多关于该函数的信息，请访问https://codex.wordpress.org/Function_Reference/wp_list_cats。

categoriesCont div标签的最终效果如下图所示：



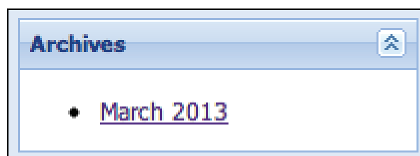
接下来声明Archives（归档）小部件：

```
<div id="archivesCont">
  <ul class="list-archives">
    <?php wp_get_archives('type=monthly'); ?>
  </ul>
</div>
```



我们可通过WordPress的wp_get_archives函数获取归档列表。若希望了解更多关于该函数的信息，请访问http://codex.wordpress.org/Function_Reference/wp_get_archives。

archivesCont div标签的最终效果如下图所示：



接下来实现Tag Cloud（标签云）小部件：

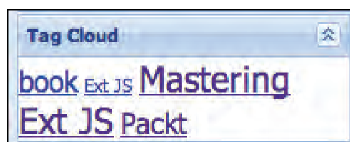
```
<div id="tagsCont">
  <?php
    $args = array(
      'smallest'  => 8,
      'largest'   => 16
     );

    wp_tag_cloud($args);
  ?>
</div>
```



我们可通过WordPress的wp_tag_cloud函数获取标签列表。若希望了解更多关于该函数的信息，请访问http://codex.wordpress.org/Function_Reference/wp_tag_cloud。

tagsCont div标签的最终效果如下图所示：



接下来实现Recent Posts（最近文章）小部件：

```
<div id="recentCont">
  <ul class="list-archives">
    <?php
      $args = array( 'numberposts' => '5' );
      $recent_posts = wp_get_recent_posts( $args );
      foreach( $recent_posts as $recent ){
        echo '<li><a href="' .
        get_permalink($recent["ID"]) .
        '" title="Look '.
        esc_attr($recent["post_title"]).'" >' .
        $recent["post_title"].'</a> </li> ';
      }
    ?>
  </ul>
</div>
```



我们可通过WordPress的wp_get_recent_posts函数获取最近文章列表。若希望了解更多关于该函数的信息，请访问http://codex.wordpress.org/Function_Reference/wp_get_recent_posts。

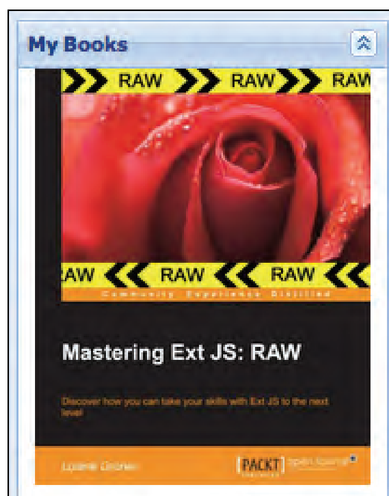
recentCont div标签的最终效果如下图所示：



你也可以在sidebar.php文件里用HTML代码创建自定义小部件。比如，可以在以下小部件里呈现一个图片。

```
<div id="booksCont">
  <!-- random HTML code -->
</div>
```

booksContdiv标签的最终效果如下图所示：

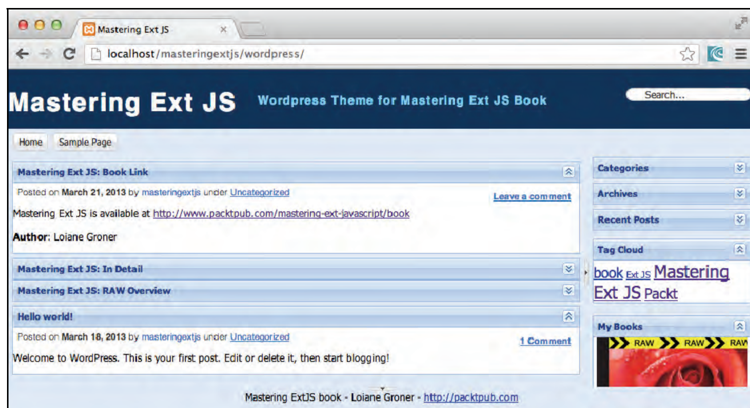


以上就是sidebar.php的代码实现。回到app.js文件里，我们需要在屏幕右侧区域添加一个表示侧边栏的新容器：

```
{
    region: 'east',
    xtype: 'container',
    collapsible: true,
    autoScroll: true,
    styleHtmlContent: true,
    layout: {
        type: 'vbox',
        align: 'stretch'
    },
    split: true,
    width: 200,
    defaults: {
        xtype: 'panel',
        padding: '5px',
        margins: '0 0 5 0',
        collapsible: true,
        styleHtmlContent: true,
        autoScroll: true
    },
    items: [{
        title: 'Categories',
        html: Ext.getDom('categoriesCont').innerHTML
    },{
        title: 'Archives',
        html: Ext.getDom('archivesCont').innerHTML
    },{
        title: 'Recent Posts',
        html: Ext.getDom('recentCont').innerHTML
    },{
        title: 'Tag Cloud',
        html: Ext.getDom('tagsCont').innerHTML
    },{
        title: 'My Books',
        html: Ext.getDom('booksCont').innerHTML
    }]
}
```

在右侧区域新容器里添加的各个子项即是刚才创建的各个div标签的HTML内容。

刷新博客，结果如下图所示：



11.8 构建单一文章页面

我们用Ext JS创建第一个属于自己的WordPress主题，现在到最后阶段了。我们还剩下两项工作没有完成：实现page.php文件和single.php文件的代码。这里我们先来创建single.php文件：

```
<?php get_header(); ?>

<?php /* If there are no posts to display, such as an empty archive page */ ?>
<?php if ( ! have_posts() ) : ?>
    <h1>Not Found</h1>
      <p>Apologies, but no results were found for the requested archive. Perhaps
searching will help find a related post</p>
<?php endif; ?>
<div id="contentCont" style="display:none;">
<?php while ( have_posts() ) : the_post(); ?>

<div id='post' class="post">
  <div id="title" style="display:none;"><?php the_title(); ?></div>
  <div class="post-details">
    <div class="post-details-left">
    Posted on <strong><?php the_date(); ?></strong> by <span class="author"><?php
the_author(); ?></span> under <span class="author"><?php the_category(',
'); ?></span>
    </div>
    <div class="post-details-right">
    <?php edit_post_link('Edit', '<span class="comment-count">  ' ,
'</span>'); ?><span class="comment-count"><?php comments_popup_link('Leave a
comment', '1 Comment', '% Comments'); ?></span>
    </div>
  </div>

  <?php if ( is_archive() || is_search() ) : // Only display excerpts for archives and
search. ?>
      <?php the_excerpt(); ?>
  <?php else : ?>
      <?php the_content('Read More'); ?>
  <?php endif; ?>

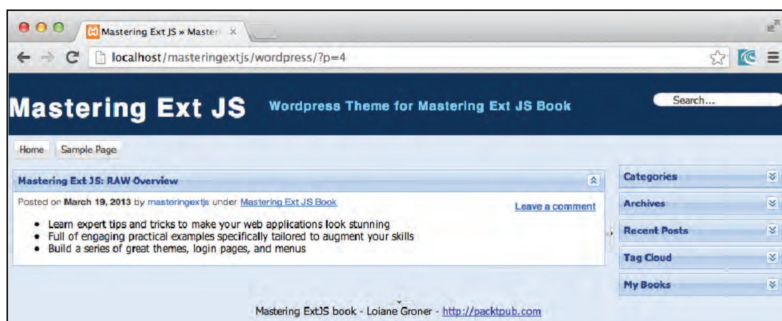
</div><!-- post -->
</div>
<div class="spacer"></div>

<?php endwhile; ?>

<div class="spacer"></div>
<?php get_footer(); ?>
```

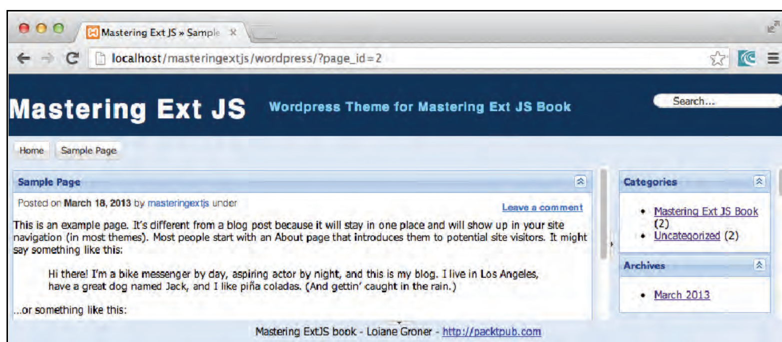
把该文件与index.php文件比较一下，你会发现两者很相似。但不同于index.php呈现所有文章的列表，WordPress在这里只取回一个文章。由于前面已经实现了Ext JS代码（共用列出所有文章的Ext JS代码），因此这里就不需要再针对Ext JS代码做什么了。

点击某个文章时，single.php执行结果如下图所示：



11.9　构建单一页面

single.php文件与page.php文件具有相同的内容。因此我们只需要复制粘贴single.php文件的内容到page.php文件即可。页面和文章在WordPress里都按同样方式处理。在博客里打开一个页面，结果如下图所示：



就这样，我们用Ext JS完成了第一个WordPress主题。接下来按照你的喜好去完善它吧。

11.10　小结

本章我们掌握了如何通过Ext JS构建一个完整的WordPress主题，了解到构建最简单的（WordPress）主题只需非常少的Ext JS代码。WordPress有非常丰富的函数供我们直接使用，使我们的工作变得更简单了。

下一章将讲解如何测试Ext JS应用程序，学习重用代码构建移动应用的方法，并告知可获取更多相关资源的地址，从而进一步丰富我们的应用程序。

第 12 章

调试与测试

本章的标题是调试与测试，但我们要讲的远不止这些。在进一步展开之前，我们先来谈谈调试技术，调试技术与编码技术同等重要。人们写代码时常常想当然地认为代码一执行就能成功，但结果往往并非如此。编写代码，然后程序抛出异常或错误，这时候，需要再次深入代码找出问题所在，这个过程是开发人员工作的一部分，甚至是生命的一部分。本章讨论的第二个话题是测试，你怎么测试一个Ext JS应用程序？编码，然后打开浏览器进行测试，甚至不借助一些方式方法。是不是还有更好的测试方式呢？当然有，而且你在本章就会看到。再讲得远一点，我们都喜爱文本编辑器（可以肯定总有一款是你钟爱的，比如Sublime Text、Text Mate、Notepad++、Eclipse、Aptana、Visual Studio、Vim，等等），但仍有很多其他工具能够帮助我们提高Ext JS开发效率，我们也将讨论它们。我们已经领略了Sencha API的惊艳效果，但世界各地的众多开发者们仍在分享他们的成果并给予我们更多帮助。其中有些功能甚至是Ext JS无法提供的，我们可以借助这些成果完善提升应用程序。最后，我们将探讨移动应用程序的开发，Ext JS框架可用来开发Web桌面应用程序，但不适用于开发移动应用程序，那么，要怎样才能使同一个应用程序在桌面和移动设备上都正常运行呢？届时我们会谈到这个话题。

本章主要包括以下内容：

- 调试Ext JS应用程序；
- 测试Ext JS应用程序；
- 有用的工具箱；
- 转换Ext JS应用为移动应用程序；
- 发现更多的开源插件。

12.1 调试 Ext JS 应用程序

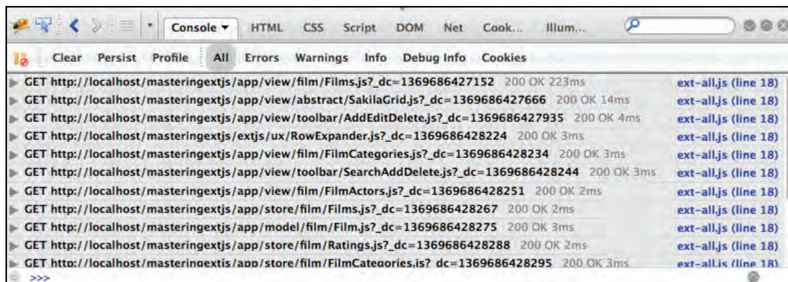
通过本书的学习，我们认识到调试的重要性，特别是当我们掌握了一种更简单的方法计算出正确`ComponentQuery`选择器时。我们用Ext JS开发应用程序时，必须使用一款调试工具。在这一过程中，我们不仅能达到调试的目的，也将掌握更多的Ext JS知识，这是一个非常好的练习机会。

我们创建Ext JS应用程序时，必须记住几件事情（不只是Ext JS，通常的JavaScript应用也如此）：大小写敏感问题；`LoginScreen`类不同于`Login screen`；注意保留字（http://mattsnider.com/reserved-words-in-javascript/），你可以将其用于命名空间、类名和包名，但不能用于变量名；拼写检查，这点非常重要，我们敲字时，有时候可能多敲了字符（胖手指综合征）。

10年前进行JavaScript编程时，我们唯一的调试工具就是`alert`函数。那时通常放几个`alert`函数在代码中，然后执行代码，看看哪个`alert`没有执行，从而找出错误所在。现在，我们有了控制台，如我们所愿可以通过控制台记录日志来发现警告和错误信息。

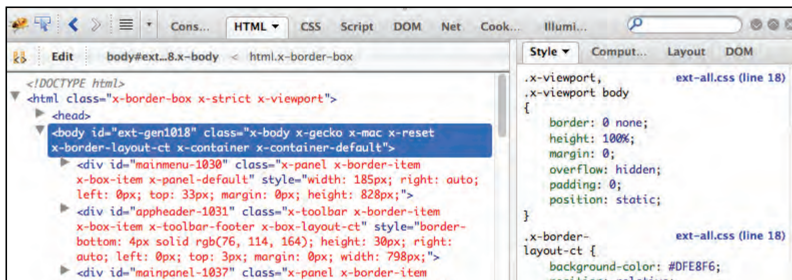
我们还有很多好的调试工具。最重要的两个是Google developer tools和Firebug for Firefox，你至少得掌握好其中一种的使用方法（它俩很相似）。

比如使用Firebug for Firefox。它有几个标签页：在Console标签页，可以查看控制台信息以及加载的文件，如下图所示：



关于文件加载成功与否，这可是个大工程。如：类名（MVC方式下）、CSS路径设置以及包含在index.html文件里的JavaScript问题，这些简单的错误都有可能导致文件加载不成功。所有这些常见错误都可以通过Console或Net标签页查清楚。

HTML标签页里可以查看详细信息，Ext JS生成的HTML代码如下：



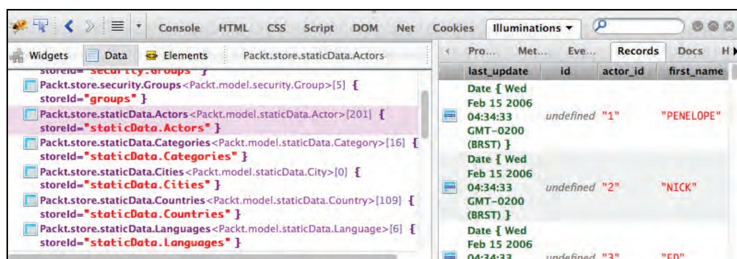
鼠标悬停时，涉及HTML代码的部分会高亮显示。还有CSS和Script两个标签页。我们可以通过实时改变CSS和脚本来观察实时效果。这可真神奇！因此，学习使用调试工具是很重要的。



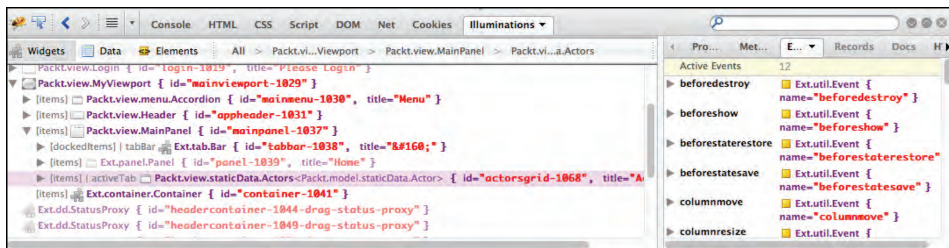
> 若希望了解更多关于Firebug的信息，请访问http://getfirebug.com/；若希望了解更多关于Google developer tools的信息，请访问https://developers.google.com/chrome-developer-tools/。

当然，如果你希望将Ext JS代码调试能力提升至一个新的高度，还有一个很特别的工具：Illuminations for developers（http://www.illuminations-for-developers.com/），这是个Ext JS专用调试工具，也是个付费工具，但性价比很高，是个Firebug/Firefox扩展。

来看看Illuminations for developers的Data标签页：



我们可以看到所有的存储器、已加载数据，以及可调用的方法（如果需要的话）。同时，也能在部件层次中看到所有的小部件及触发事件：



精通调试工具与精通Ext JS编程技术同等重要。选择你喜爱的工具，体验编程和调试的乐趣吧！

12.2　测试 Ext JS 应用程序

测试是应用开发及维护工作的重要组成部分。如果不编写测试代码，我们就只好手工测试每个使用案例，而一旦我们改动了代码，就不得不重新再手工测试一遍。相同的情况也会发生在我们维护代码的过程中。开发人员通常只测试改动部分的代码，但正确的方式应该是进行回归测试以搞清楚改动部分是否会破坏其他部分。因此，花些时间写测试代码将大大节省后期时间。你在前期会花多一点的时间，但是，之后你却可以做到点击一下，就能够运行所有的测试并验证什么被破坏了、什么还是正常的。

我们还经常对服务器端代码做单元测试，Java、PHP、Ruby和C#社区提供了许多可以对服务器端代码进行单元测试的工具，但有时候我们会忘了对前端代码进行单元测试（这里指的是Ext JS）。不用担心，也有些工具可以用来测试Ext JS代码。

一个非常流行的JavaScript测试工具就是Jasmine（http://pivotal.github.io/jasmine/）。Jasmine是一个用于行为驱动开发（Behavior-driven development，BDD，http://en.wikipedia.org/wiki/Behavior-driven_development）的测试工具。在Ext JS文档中可以找到两处用Jasmine进行Ext JS应用测试的参考：http://docs.sencha.com/extjs/4.2.0/#!/guide/testing 和 http://docs.sencha.com/extjs/4.2.0/#!/guide/testing_controllers。

我们介绍的例子使用另一个名为Siesta的测试工具（http://www.bryntum.com/products/siesta/）。Siesta也能用于测试JavaScript代码，但其最精彩的部分是提供了测试Ext JS应用程序的专用API。

开始之前，列一下测试Ext JS应用程序的操作步骤：

(1) 使用Sencha command生成"测试"构造；
(2) 测试"测试"构造；
(3) 安装Siesta；
(4) 创建harness（test harness：测试辅件，测试夹具）；
(5) 创建测试用例。

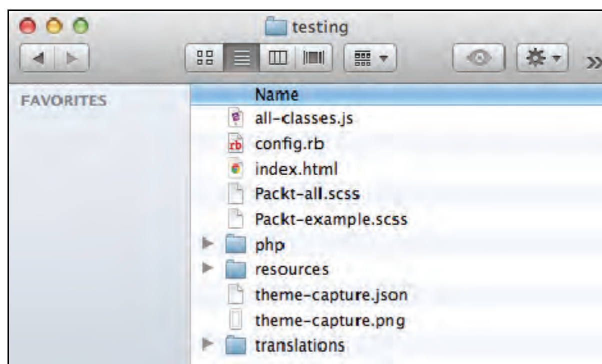
让我们开始吧！

12.2.1　使用Sencha command生成"测试"构造

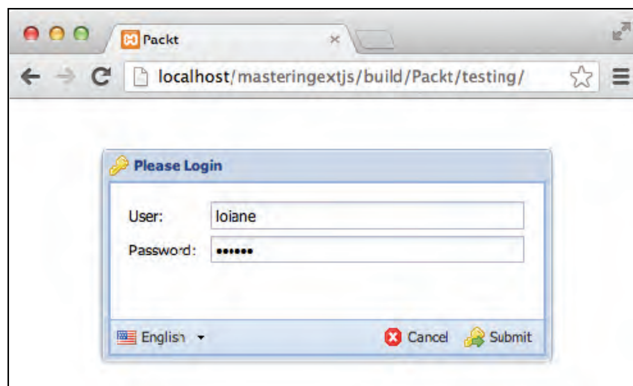
第一步是使用Sencha command生成项目的测试构造。我们在第10章已经学习了操作方法，打开终端应用程序，切换到应用程序目录，并执行`sencha app build testing`，截图如下所示：

```
masteringextjs — java — 80×24
loiane:~ loiane$ cd /Applications/XAMPP/xamppfiles/htdocs/masteringextjs
loiane:masteringextjs loiane$ sencha app build testing
Sencha Cmd v3.1.0.256
[INF] Including theme package ext-theme-classic for app.theme=ext-theme-classic
build
[INF]
[INF] init-plugin:
[INF]
[INF] init-plugin:
[INF] Invoking plugin (/Applications/XAMPP/xamppfiles/htdocs/masteringextjs/.sen
cha/app/plugin.xml) - supported targets: -before-app-build
[INF]
[INF] -before-app-build:
[INF] Invoking plugin (/Applications/XAMPP/xamppfiles/htdocs/masteringextjs/.sen
cha/app/plugin.xml) - supported targets: app-build
[INF]
[INF] cmd-root-plugin.init-properties:
[INF]
[INF] init-properties:
[INF]
[INF] init-sencha-command:
[INF]
[INF] init:
[INF]
```

Sencha command将在masteringextjs/build/Packt/testing目录中创建一个用于测试的新文件夹，其中还包含了测试用的编译代码。别忘了复制php和translations文件夹到testing文件夹中；否则，测试构造过程就会出错：



下一步是测试“测试”构造。要达成此目的，需执行链接http://localhost/masteringextjs/build/Packt/testing,如下图所示：



一切正常。

12.2.2　安装Siesta并创建测试用例

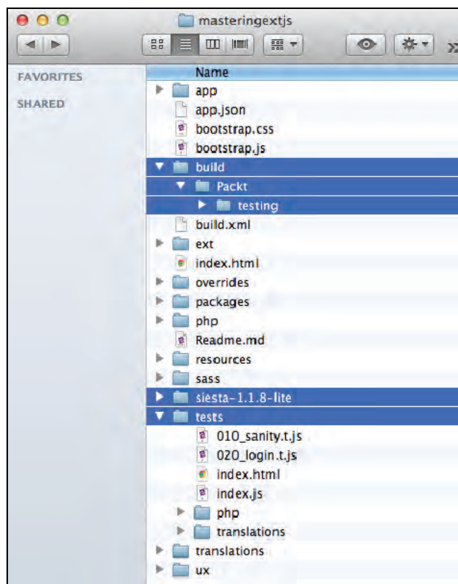
接下来我们安装Siesta，以便可以用这个好工具创建测试用例。

首先，从http://www.bryntum.com/products/siesta/下载Siesta。对于我们的示例，使用免费的Lite版（精简版）即可。付费版包含服务支持、Selenium和Phantom JS集成以及跨页测试等增值内容。

下载Siesta后，将其解压至项目文件夹中。在我们即将开始编写测试代码之前，先回顾一下还需要做些什么：

(1) 创建tests文件夹；
(2) 创建Harness文件；
(3) 创建index.html文件；
(4) 创建各个所需测试用例。

因此，第一步是在项目文件夹中创建一个名为tests的新文件夹。本节结束后，效果如下图所示：



接下来创建Harness文件。我们希望把所有测试用例都看成是项目的一部分，在Harness文件中声明这些测试用例。将Harness文件命名为index.js，并在tests文件夹中创建它，文件内容如下：

```
var Harness = Siesta.Harness.Browser.ExtJS;

Harness.configure({
    title       : 'Mastering Ext JS Test Suite',

    preload     : [
        '../build/Packt/testing/resources/Packt-all.css',
        '../build/Packt/testing/resources/css/app.css',

        '../build/Packt/testing/translations/locale.js',
        '../build/Packt/testing/all-classes.js'
    ]
});

Harness.start(
    '010_sanity.t.js',
    '020_login.t.js'
);
```

我们在Harness文件的一处配置两个测试用例：`010_sanity.t`和`020_login.t`。同时还需要在`preload`属性配置项中添加我们的CSS及应用程序JavaScript等相关文件；程序代码若想能正常测试，就要让Siesta预加载应用程序的CSS以及JavaScript文件。

下一步创建tests/index.html文件：

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" type="text/css" href="../bootstrap.css">
        <link rel="stylesheet" type="text/css" href="../siesta-1.1.8-lite/
resources/css/siesta-all.css">

        <script type="text/javascript" src="../ext/ext-all.js"></script>
        <script type="text/javascript" src="../siesta-1.1.8-lite/siesta-all.js">
</script>
         <script type="text/javascript" src="translations/locale.js"></script>
        <script type="text/javascript" src="index.js"></script>
    </head>

    <body>
    </body>
</html>
```

在这段HTML代码中，我们导入了Siesta的JavaScript和CSS文件，Ext JS及其CSS文件，以及前面创建的Harness文件index.js。

现在开始创建项目需要的所有测试用例（这里是两个）。首先在tests文件夹中创建第一个测试用例文件010_sanity.t.js：

```
StartTest(function(t) {
    t.diag("Sanity");

    t.ok(Ext, 'ExtJS is here');
    t.ok(Ext.Window, 'Ext.Window as well');

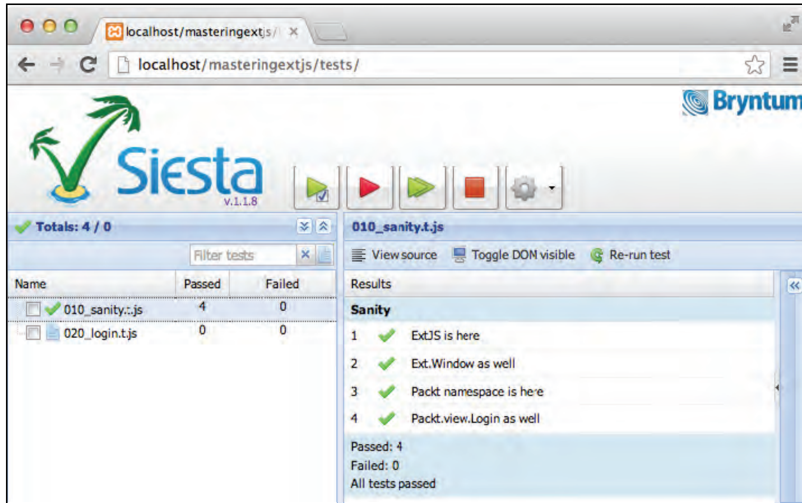

    t.ok(Packt, 'Packt namespace is here');
    t.ok(Packt.view.Login, 'Packt.view.Login as well');

    t.done();
});
```

我们要理解上述代码。所有通过Siesta实现的测试代码必须都放在`StartTest(function(t)`构造函数里。

接下来，在构造函数里加入了一些断言代码。第一个测试用例是一个很简单的示例：确认某些类已加载，比如Ext命名空间以及应用程序命名空间。之后，我们可以确认Ext JS框架和我们应用的任意类已存在，这样就可以开始一些更复杂的测试了。

现在，尝试执行第一个测试用例。要达成此目的，请在浏览器中访问：http://localhost/masteringextjs/tests。测试结果如下图所示：



现在我们来创建第二个测试用例。在tests文件夹中创建020_login.t.js文件：

```
StartTest(function(t) {
    t.diag("Sanity test, loading classes on demand and verifying they were indeed
loaded.");

    t.ok(Ext, 'ExtJS is here'); // #1
    t.ok(Packt, 'Packt namespace is here'); // #2

    t.requireOk('Packt.view.Login'); // #3

    t.waitForComponent('Packt.view.Login', true, function(){// #4

          var submitButton = Ext.ComponentQuery.query('login button#submit')[0];

          t.chain(
              { action : 'click', target : submitButton } // #5
          );
          t.waitForComponent('Packt.view.MyViewport', true, function(){  // #6
                t.ok(Packt.view.MyViewport, "Packt.view.MyViewport was rendered."); // #7
            t.done();
        });
    });
});
```

观察一下代码，我们会发现这个测试用例变得复杂了。总体而言，这个测试用例确认Ext（#1）和Packt（#2）命名空间已加载，之后请求加载Packt.view.Login类（#3），然后等待，直到组件在屏幕上渲染呈现（#4：等待加载页面淡出）。一旦登录界面渲染出来，测试用例就会按下

提交按钮（#5：出于测试目的，用户名和密码已经在对应文本字段组件中预设了）。一旦提交按钮被点击，测试套件将等待，直到视见区渲染呈现（#6），然后断言宣告类已加载（#7）。

Siesta测试框架的一个最优特性就是它具有针对Ext JS框架的特定测试用例，从而使测试相对其他工具而言容易得多。另一个优点是测试构造，也就是说测试用到的业务逻辑代码仍是我们应用程序中的代码，不需要在测试用例中再写一遍，只需如上述实现的测试用例那样简单地调用这些代码即可。

如希望了解更多关于的Siesta断言功能及测试能力的信息，请参考：http://www.bryntum.com/docs/siesta/，或者访问Siesta安装目录下的docs文件夹。

12.3　有用的工具箱

这里我们将介绍一些能够给Ext JS应用程序开发带来很大帮助的工具。本节后面会列出所有这些工具的链接地址。

第一个工具是JSLint，它能够帮助我们找出Java错误，并清理代码。

第二个工具是YSlow。YSlow分析Web页面并告知我们：基于高性能网站的规则标准，为什么这些页面是低性能的。YSlow是一个集成在流行的Firebug Web开发工具里的Firefox扩展。

Ext JS是一个JavaScript框架，JavaScript性能是许多开发团队关心的话题。最理想的状态是用户在浏览器中尽可能少地加载JavaScript代码。这也是使用Sencha command进行产品构造，而不能简单地部署所有应用文件到生产环境的重要原因。

Sencha command也能把Ext JS的CSS文件体积缩减到更小，我们还可以只包含真正能用到的组件的CSS（比如创建一个自定义主题）。但是，通常情况下，会有一个声明了图标以及自定义样式的应用程序的CSS。在这里，我们有三个非常有用的小技巧来针对现实情况进行优化。对于自定义CSS，可以使用如Sass或Less这样的预处理器。假设我们用Sass或Less创建`app.css`，那么要记住编译输出的CSS体积已经缩减了。第二个小技巧：即使不使用Sass或Less编译自定义CSS，而是使用像YUI compressor和CSS minifier这样的工具，也能够达到在生产环境上部署缩减版CSS的目标。第三个小技巧：通常情况下，会有多个自定义CSS文件，这样可以更好地组织样式。比如，一个CCS文件专门设置图标，另一个针对普通设置，再有一个又针对另一特定目的。但对于生产环境而言，始终要记得把它们合并成一个CSS文件，这样在浏览器加载时就会更高效。有一个叫grunt-contrib-concat的工具可以帮助我们达成此目的。

CSS精灵（CSS Sprite，CSS图像拼合技术），是另一个非常重要的话题。还记得在应用程序的很多地方都用到的图标吗？还记得按钮图标、面板图标及标签页图标吗？你可以用该图像拼合方法创建一张带有所有图标的CSS精灵图片。在CSS里，我们只需加入这个单一图片，按照如下

代码，设置对应图标的background-position：

```
.icon-message {
  background-image: url('mySprite.png');
  background-position: -10px -10px;
}

.icon-envolope {
  background-image: url('mySprite.png');
  background-position: -15px -15px;
}
```

还有许多的工具可以用来创建CSS精灵图片：SpritePad、SpriteMe以及Compass Sprite Generator等。

这里提到的所有工具，其参考链接如下。

- JSLint　http://www.jslint.com/
- YSlow　http://developer.yahoo.com/yslow/
- Sass　http://sass-lang.com/
- Less　http://lesscss.org/
- YUI Compressor　http://developer.yahoo.com/yui/compressor/
- CSS Minifier　http://www.cssminifier.com/
- grunt-contrib-concat　https://npmjs.org/package/grunt-contrib-concat
- SpritePad　http://wearekiss.com/spritepad
- SpriteMe　http://www.spriteme.org/
- Compass Sprite Generator　http://compass-style.org/help/tutorials/spriting/

别忘了Ext JS也是JavaScript，因此也需要关注其性能。通过实践上述小技巧，Ext JS应用程序的性能同样能够得到提升。

最后，有两个来自Sencha的重要工具：Sencha Architect和Sencha Eclipse plugin。Sencha Architect是一个类似Visual Studio的可视化设计工具：可拖放并能够看到应用程序的样子，所有属性配置项都要通过配置面板设置，只有方法、函数及模板可以进入并编码。Sencha Architect的优点是能够帮助我们按最佳实践开发程序，其生成的代码组织得非常好。你可以用Sencha Architect开发所有的Ext JS程序，而对于服务器端代码，你仍然可以使用你最喜爱的IDE来开发（Eclipse、Aptana、Visual Studio或其他工具）。

Sencha Eclipse Plugin是一个Eclipse插件，具备代码自动完成功能。Sencha Architect和Sencha Eclipse plugin都是付费工具。但你可以出于测试目的下载其试用版本：http://www.sencha.com/products/complete/、http://www.sencha.com/products/architect/。

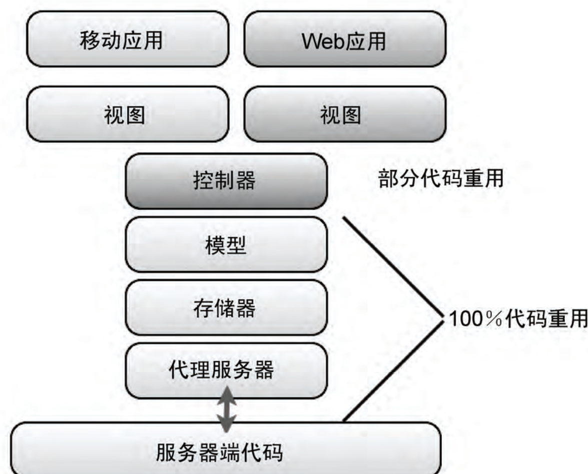
12.4　从 Ext JS 应用到移动应用

移动应用程序当下非常流行。假设我们开发出了一个很好的Ext JS应用程序，通常我们希望在移动设备上也能够有个同样的应用。大多数人都会认为Ext JS技术是跨浏览器的，只需要把应用程序发布到互联网，然后再把链接发给移动用户就万事大吉了。但事实上并非如此。虽然Ext JS也兼容移动浏览器，但它并不能为移动用户提供最佳的用户体验。如果希望为用户提供极佳的移动体验，就需要把Ext JS应用转换为移动应用。但是，要怎么做呢？学习Objective-C、Android Java开发或是其他的语言又得花不少时间，而我们也不太可能总有那么多时间。

因此，这里打算介绍一下Ext JS的“表兄弟”——Sencha Touch。Sencha Touch是市场上首款HTML5移动开发框架。另一个好消息是：不需要重写所有的代码就能让应用程序适配移动设备。

Sencha Touch跟Ext JS共用一套API。数据包（指`Ext.data`）如模型、存储器以及框架核心，都是共用一个。Sencha Touch同样也是使用MVC架构。控制器和视图（组件）的应用方式与Ext JS也非常类似。当然，最大的不同点在于视图，毕竟Web组件不同于移动组件。然而，Sencha Touch也提供表单、列表、甚至为移动设备定制的网格组件等内容。

至于使用Sencha Touch可以重用多少代码量，下图给出了说明：



所以，能够复用的代码量是非常大的。同时，Sencha Touch移动应用有两种部署方式：第一种是Web App方式，移动应用部署在站点上，用户访问Sencha Touch部署的链接地址（Sencha Touch跟服务器端代码都在同一个域上）；第二种是混搭方式，Sencha Touch应用部署在用户的移动设备上（Sencha Touch提供了iOS及Android的原生打包工具，但也可以生成Sencha Touch的原生Blackberry 10和Windows Phone 8应用），服务器端代码部署在服务器或Web上。这种情况下，可

以通过CORS来实现应用与服务器端代码间的通信（我们已经在第10章讨论过了）。



如希望了解更多关于Sencha Touch的内容，请参考http://www.sencha.com/products/touch/。

12.5　第三方组件和插件

尽管Ext JS提供了大量的组件，但我们还是有可能需要开发自己的组件，或者使用其他开发者开发的组件。关于该主题的Ext JS社区规模非常大。大量开发者在社区里分享他们的组件、扩展以及插件。以下是两个主要社区及地址。

- Sencha市场　https://market.sencha.com/
- Sencha论坛　http://www.sencha.com/forum/forumdisplay.php?82-Ext-User-Extensions-and-Plugins

12.6　小结

本章我们了解了调试Ext JS应用程序的重要性及相关知识，还了解了一些能够给我们的调试工作带来帮助的工具。我们学习了如何使用Siesta（开源工具，提供了精简版及付费版）创建Ext JS应用程序的测试用例。同时，了解了性能优化的重要性，并通过一些免费工具优化了我们的Ext JS应用程序的性能。最后我么还掌握了一些如何把Ext JS应用程序迁移到移动平台上的方法，并知道在哪里可以找到项目所需的插件、扩展以及新组件。

现在，让我们充分发挥创造力，使用Ext JS来创建了不起的应用吧！

关注图灵教育 关注图灵社区

iTuring.cn

在线出版　电子书《码农》杂志　图灵访谈 ……



QQ联系我们

读者QQ群：218139230



微博联系我们

官方账号：@图灵教育 @图灵社区 @图灵新知

市场合作：@图灵袁野 @图灵刘紫凤

写作本版书：@图灵小花 @陈冰_图书出版人

翻译英文书：@李松峰 @朱巍ituring @楼伟珊

翻译日文书或文章：@图灵乐馨

翻译韩文书：@图灵陈曦

电子书合作：@hi_jeanne

图灵访谈/《码农》杂志：@李盼ituring

加入我们：@王子是好人



微信联系我们



图灵教育
turingbooks



图灵访谈
ituring_interview

“总体来说，面对Ext JS和JavaScript我都是入门者，但《精通Ext JS》让我将已有知识拼接融合，提升了自己的编程技能。本书每章介绍一个功能模块，带我们渐进式开发应用，非常值得一读。”

“《精通Ext JS》是至今为止介绍Ext JS的难得佳作。首次揭秘Ext JS 4.2的开发技巧……作者还探讨了构建Ext JS应用的客户端逻辑和服务器端逻辑及一些MySQL知识。值得一提的是，其中采用MVC式开发方法讲述Ext JS应用开发，而这正是Sencha文档中多数示例所欠缺的。”

——亚马逊读者评论

Ext JS是一个用JavaScript编写的、独立于后台技术的前端AJAX框架，可以用在.NET、Java、PHP等各种编程语言开发的应用中，以开发华丽的富客户端应用。用Ext JS打造的RIA Web应用不仅具有与桌面程序一样的标准用户界面与操作方式，而且能够跨浏览器平台运行。Ext JS业已成为开发具有完满用户体验的Web应用的完美之选。

作为一本内容详实的Ext JS学习指南，《精通Ext JS》以Ext JS 4.2为依托，站在开发者的角度思考问题，将应用划分为不同的功能模块，一章解决一个任务，带我们渐进式开发基于MVC的完整应用，经历从界面原型到产品上线前的各个阶段。其中，你将学会实现用户及分组安全功能，掌握网格、表单、图表和树形结构，以及将不同表示结构的内容导出成PDF和Excel格式的最佳实践方式。而且，在开发完成应用程序的所有功能后，Loiane Groner还将带你自定义漂亮的主题，以突出应用程序的个性化风格，为产品上线做足准备。

通读本书，融会贯通Ext JS应用开发的各种知识与思路，你将能够用Ext JS创建绝妙的应用！

主要内容：

- 开发内容管理模块；
- 服务器端的信息处理（避免使用JSON文件）；
- 构建WordPress主题（Ext JS的不同应用场景）；
- 开发电子邮件客户端、分组及安全模块；
- 构建产品级应用；
- Ext JS应用调试与测试；
- 重用代码构建移动应用的方法。



[PACKT]
PUBLISHING

图灵社区：iTuring.cn
热线：(010)51095186转600

分类建议 计算机/程序设计/Ext JS

人民邮电出版社网址：www.ptpress.com.cn



ISBN 978-7-115-34723-7
定价：59.00元

欢迎加入

图灵社区

电子书发售平台

电子出版的时代已经来临，在许多出版界同行还在犹豫彷徨的时候，图灵社区已经采取实际行动拥抱这个出版业巨变。相比纸质书，电子书具有许多明显的优势。它不仅发布快，更新容易，而且尽可能采用了彩色图片（即使有的书纸质版是黑白印刷的）。读者还可以方便地进行搜索、剪贴、复制和打印。

图灵社区进一步把传统出版流程与电子出版业务紧密结合，目前已实现作译者网上交稿、编辑网上审稿、按章发布的电子出版模式。这种新的出版模式，我们称之为“敏捷出版”，它可以让读者以较快的速度了解到国外最新技术图书的内容，弥补以往翻译版技术书“出版即过时”的缺憾。同时，敏捷出版使得作、译、编、读的交流更为方便，可以提前消灭书稿中的错误，最大程度地保证图书出版的质量。

开放出版平台

图灵社区向读者开放在线写作功能，协助你实现自出版的梦想。你可以联合二三好友共同创作一部技术参考书，以免费或收费的形式提供给读者，这极大地降低了出版的门槛。成熟的书稿，有机会入选出版计划，同时出版纸质书。

图灵社区引进出版的外文图书，都将在立项后马上在社区公布。如果有意翻译哪本图书，欢迎来社区申请。只要通过试译的考验，即可签约成为图灵的译者。当然，要想成功地完成一本书的翻译工作，是需要有坚强的毅力的。

读者交流平台

在图灵社区，读者可以十分方便地写作文章、提交勘误、发表评论，以各种方式与作译者、编辑人员和其他读者进行交流互动。提交勘误还能够获赠社区银子。欢迎大家积极参与社区开展的访谈、审读、评选等多种活动，赢取银子，可以换书哦！